Diseño e implantación de una aplicación móvil de control remoto para un vehículo

TRABAJO FIN DE GRADO

UNIVERSIDAD CARLOS III DE MADRID
Ingeniería informática

Madrid, septiembre 2017

Autor: Luis Alberto Pantaleón del Puerto

Tutor: Javier Fernández Muñoz

Abstract

In this chapter we will try to summarize the project, beginning with what was the motivation to select the topic of the project, its overview and the objectives that we are trying to reach by making it.

Motivation

At the present time, we can see how unmanned vehicles, controlled remotely or automatic, have had a big boom as the technology has advanced enough to make them possible. Since professional to leisure area, we can see how its possibilities open a new exploitable market.

In view of this fact, we was worried about if it would be possible, in an academic area, to develop an application that controls a vehicle remotely on the one hand and automatically in the other hand. So this project will try to reach this objective, using a car replica as the vehicle.

Therefore, the main motivations of this project are:

- Integrate in one unique application the both control modes for the vehicle: automatic and remote.
  - Automatic Mode. The user will can determine a set of movements, so the vehicle will determine the path to follow using a camera as unique sensor.
  - Remote Mode. The user will can determine the next movement of the vehicle anytime.
- Provide an application for mobile devices whose user interface (UI) is clear and simple and, by using it, the user will can select the mode and the movements (in case of the remote mode). In both modes, the user will can see the environment from the vehicle perspective through the camera said before.
- The vehicle must be as autonomous as possible as far as the processing, so the mobile application will be the only component that is not physically into the vehicle itself.

Overview

The project is about the design and implantation of a mobile application that allows the remote control of a vehicle (car replica), providing two different control modes:

- **Manual control**. The user controls the vehicle directly, specifying all the time which will be the next movement of the car.
- **Automatic control**. The user controls the vehicle indirectly, specifying only the set of movements that they want it to do, so the application will decide when to do this movements.

To develop the application, we will divide it in 3 modules that will interact with each other's:

- **Mobile application**. Component that is executed on the mobile device and is the one that provides the UI that allows to control the vehicle remotely. In this interface users will can select the control mode and the movement set. All this information will be transferred to the *Server*.
- **Server**. Component that collects the information from the *Mobile Application* and acts in consequence. If the mode selected is manual, the component will allow the direct control of the vehicle from the *Mobile Application*. On the other hand, if the mode selected is the automatic one, the component will take the movements from the user set and it will carry them out when it decides. Therefore, all the movements that the vehicle must do (in both modes) will be sent to the *Controller* component.
- **Controller**. Component that is executed over the vehicle device, in direct contact with the actuators. It will collect the movement requests from the server and it will interact with the actuators to do the movement required.

Objectives

The main objective of the project is to create an application that allows users, using a mobile device, to control a vehicle remotely or to specify a set of movements that the vehicle will carry out autonomously.

To reach the main objective, we should also have to achieve some other sub objectives that are:

- The UI provided by the *Mobile Application* (in both modes) must be intuitive and easy to use.
- Users will can see the vehicle environment from the vehicle perspective using the *Mobile Application*.
- Mobile Application must allow to store persistently a journey (set of movements) to be executed later. Users will can edit or delete them using the application.
- The *Server* must be executed over a hardware that supports a camera connection, as well as the capture, processing and sending the pictures from that camera.
- The *Server* will be in charge to capture, process and send the pictures to the application, either for the environment overview or the automatic mode.
- The *Server* will only allow one simultaneous connection, so we will be sure that there is only one user operating the vehicle.
- The *Controller* must provide a reusable (for both modes) interface for the movement requests.
- The application will can change the control mode without reset it or change a component.
- Every component will have a proper way to act in case of errors, so the application won't close abruptly.

Results and conclusions of the project

Considering the objectives established in 1.3. OBJETIVOS, the conclusions that we have obtained are:

- **The main objective has been reached** as well as the implementation of all its components: "*To develop an application that allows a user to control a vehicle remotely and automatically through a mobile application*".
- **The Mobile Application component has been developed** for Android devices. It is used as an interface so the user can interact with the rest of the application. It offers the possibility to select between the two control modes, showing the corresponding UI for each of these modes.
    - As we indicated in 4.3 Interface Design, the component provides a dedicated interface to each control mode, which has been designed to be clear and intuitive for the user.
    - Through Webviews in the graphical interface of the *Mobile Application*, the images transmitted by the *Server* from the camera of the vehicle can be observed while either the automatic or manual mode is working. These Webviews are connected to an HTTP address where those images are transmitted by the *Server*.
    - The application implements a SQLite database for path storage. In addition, graphical interfaces are offered in the Mobile Application to be able to perform in a simply way the creation, deletion and modification of these paths (the user doesn't know if there is a database or how it works).
- **The *Server* component has been developed**. It performs the intermediate layer functions between the *Controller* and the *Mobile Application* in the manual control mode. In automatic control mode, it makes autonomous driving decisions. This has been implemented in C++ and runs on the vehicle in a Raspberry Pi 3 Model B. In this component the following sub-objectives have been carried out:
    - The hardware used is, as said, a Raspberry Pi 3 Model B in addition to a camera connected to it.
    - The board can capture, transmit and store images. It uses the installable functionality in Linux OS called *Motion*. This allows you to perform:
        - Continuous capture and storage of the images.
        - Set up a server for the transfer of images to an address of type HTTP.
    - It can manage and process the images by using the free *OpenCV* image processing library.
    - The decisions of the automatic driving are made by the images captured and stored by *Motion*.
    - Only a user is allowed to connect. The rest of the connections are ignored until the requests of the actual connected user are finished.

- **The Controller component has been developed**. It is responsible for interacting with the actuators and has been implemented in an Arduino UNO with the Adafruit Moto/Stepper/Servo Shield v1.2. In this component the following sub-objectives have been carried out:
    - It provides a request and response interface for the realization of movements, which can be used in both control modes.
    - The request used in both modes are the same, so the component only has one interface.
- It is not necessary to reset any component to change the control mode. The *Mobile Application* is in charge to notify this changes.
- Error control is performed on all components, so that the system is tolerant to the errors that the user can perform using the Mobile Application or due to network problems.

Therefore we can concluded that all the established objectives have been fulfilled in a correct way. At the end of the project, we offer a prototype application that meets all the objectives initially marked.

Most of the problems found have been for lack of performance, considering that the Raspberry Pi has limitations of hardware power. Regarding the implementation, no outstanding problems were found, being its level of difficulty within the accepted.

Personal conclusions

I chose to carry out this project, as indicated in section 1.1 MOTIVATION, because I raised myself the concern to know and verify if with the knowledge acquired during the degree, I could carry out a project of these characteristics.

Now, once I have completed the project, I can conclude that, with the knowledge acquired during the degree, it can be carried out without great difficulties.

Although, that does not mean that the project was easy or simple. I have had to invest a lot of hours to be able to carry it out, gathering (and contrasting) information; without forgetting the studies and wide variety of knowledge in different platforms and scopes, in order to obtain optimal results.

The project has a wide degree of freedom, allowing me to carry out the implementation that I considered opportune at all times. It is not the type of project I am used to do during the degree, which has strict guidelines that close the design to a few possibilities.

As for the hardware used, it has been very interesting to integrate in a single application programs executed in different devices with very different characteristics.

As a final summary, the project has seemed very interesting and fun. In turn, I have been able to acquire, apply and develop the knowledge acquired of the platform Rasberry Pi - this had not been used previously for any project -. On the other hand, this project have unified three of my favorite hobbies: computer science, electronics and motorsports. Thus ensuring an extra motivation for dedication, effort and improvement, minimizing the obstacles and adversities that were emerging in the development of the project. Finally, it has allowed me to integrate in a single application my knowledge about Java, C and C++.

Future works

The project is an initial prototype with respect to the application, which was wanted to be carried out at a first moment but due to lack of time, complexity, cost and other inconveniences, finally it could not be carried out.

The application does not differ in excess on the original idea, perhaps in the beginning, the sections were somewhat more striking.

Next we will indicate by each of the components that make up the application, those characteristics that were intended to be realized initially, but have not been able to carry out, hoping that in the future this project can be continued and realized.

Mobile application

AlertDialog in the connection to the *Server*

On those screens that a connection to the server must be made, the dialog box generated and displayed would not be used to specify an IP. In it, a list with a set of devices should appear. These sets of devices would correspond to devices that are connected to the same local network, as the mobile device that runs the *Mobile Application*.

Therefore, it was only necessary to click on an item in the list of the dialog box, in order to try to connect to it. In addition, this dialog box would have preferred positions for those devices to which a previous connection was made.

Interface *Manual Control*

In the *Manual Control* screen, no buttons would be used to make motion requests for the vehicle or camera. In the original case, it was necessary to use joystick, which according to its displacement allowed to regulate the movement that was wanted to do, controlling the speed of said movement.

Server

Movements of Automatic Mode

The autonomous conduction, would not be realized by discrete movements; initially they were continuous.

In this case, this could not be done because due to the hardware available at the moment, it is not possible to guarantee a minimum framerate for the camera. This causes the vehicle (in certain cases) to move faster than the image capture, so the vehicle constantly goes out from the circuit.

Turns in Automatic Mode

At first, to make the vehicle turn, we wanted to specify all the directions by angles, so the input would be standard and more exact. Nevertheless, finally we only specify a direction (as right/left) during a variable period of time, so we can stop the turn in the moment the vehicle detects that it is out.

Traffic signal detector

Initially, the *Server* was going to have a traffic signal detector, which recognize signals that would appear on the images captured by the vehicle's camera and how far they are.

Once the signal is detected, the vehicle's distance from said signal will be checked. If the vehicle is at enough distance, the server would carry out the corresponding actions of the control of the vehicle to enforce the signal.

For example, if the vehicle observes a STOP signal and it is close enough, it will stop the vehicle for a few seconds.

Controller

Collision detector

The vehicle was going to have an ultrasonic sensor used to detect collisions or obstacles. The controller would be in charge of obtaining the information from the sensor.

If a possible collision is detected, the *Controller* would stop the vehicle and notify the other components via the serial port. It is necessary to do this processing in the *Controller* to carry out the reaction as fast as possible, avoiding the collision.

Índice de contenidos

Índice de tablas

Índice de Ilustraciones

1. Introducción

En este apartado se muestra una visión general del proyecto. Señalando, la motivación y los objetivos que han dado lugar al mismo.

Por último, se indicará, la estructura que sigue este documento.

1.1. Motivación

Actualmente, se ha producido un auge de vehículos no tripulados, tanto controlados de forma autónoma como de forma remota, gracias a que la tecnología ha evolucionado lo suficiente para poder llevarse a cabo este tipo de vehículos. Debido a esto, se han podido observar sus posibles aplicaciones, desde el ámbito del ocio hasta el ámbito profesional, abriendo así un nuevo mercado explotable.

A la vista de esto, nos surgió la inquietud de conocer si se podría realizar, en el ámbito académico, una aplicación que permitiera controlar un vehículo tanto de forma autónoma como de forma remota. Por ello, se decidió intentar llevar a cabo esta aplicación utilizando como vehículo una réplica de un coche.

Por lo tanto, las motivaciones principales de este proyecto son:

Integrar en una única aplicación dos modos de control para un vehículo: Un modo de control autónomo y otro modo de control remoto. El modo control remoto, debe permitir al usuario seleccionar en todo momento que movimiento debe realizar el vehículo. El modo de control autónomo debe permitir a un usuario especificar un trayecto y que el vehículo sea capaz de realizarlo de forma autónoma, especificando los distintos movimientos que se deben realizar en dicho trayecto. Además, la conducción autónoma se deberá de realizar utilizando como único sensor una cámara.

Ofrecer al usuario una aplicación para dispositivos móviles cuya interfaz sea sencilla y clara, mediante la cual, el usuario pueda especificar el modo de control deseado y que movimientos quiere que realice el vehículo. Esta aplicación debe ofrecer en los dos modos de control la posibilidad de observar el entorno en el que se encuentra el vehículo, todo ello sin que sea necesario que el usuario se encuentre en dicho entorno, para ello deberá de utilizar la cámara antes mencionada.

El vehículo debe ser lo más autónomo posible en lo que se refiere a procesamiento, siendo el dispositivo móvil que ejecuta la aplicación móvil el único que no se encuentra sobre el vehículo.

1.2. Visión general

El proyecto consiste en el diseño e implantación de una aplicación móvil que permita el control remoto de un vehículo, para ser más específicos, en este caso, el control remoto de la réplica de un automóvil.

La idea es ofrecer al usuario una aplicación que le permita controlar el vehículo desde un dispositivo móvil. Ofreciendo para ello dos modos de control sobre el mismo:

- **Control manual:** Modo mediante el cual el usuario controlará el vehículo directamente. Por lo tanto, el usuario será el que especifique en todo momento cual es el movimiento que quiere que realice el vehículo.
- **Control automático:** Modo mediante el cual el usuario controlará el vehículo indirectamente. Para llevar acabo esto, el usuario indicará de antemano cuales son los movimientos que quiere que realice el vehículo y la aplicación será la encargada de llevarlos a cabo cuando lo consideré oportuno.

La aplicación estará compuesta de tres módulos o programas independientes, que a su vez interaccionan entre ellos:

- **Aplicación móvil:** Componente que se ejecutará en un dispositivo móvil, encargado de ofrecer una interfaz al usuario que le permita controlar el vehículo de forma remota. En esta interfaz, el usuario podrá indicar el modo de control y los movimientos que quiere que realice el vehículo. Esta información se transferirá al *Servidor*.
- **Servidor:** Componente de la aplicación encargado de recoger la información referente al modo de control y los movimientos, la cual es transferida por el usuario mediante la *Aplicación móvil*.
  Este componente, realizará unas acciones u otras dependiendo del modo de control escogido por él usuario mediante la *Aplicación móvil*.
  - Si el modo de control es manual, permitirá un control directo sobre él vehículo desde la *Aplicación móvil*.
  - Por otro lado, si el modo de control es automático, tomará los movimientos especificados por el usuario desde la *Aplicación móvil,* y será el encargado de llevarlos a cabo cuando considere oportuno, realizando así una conducción autónoma del vehículo.

  Por último, todo aquel movimiento que este componente quiera que realice el vehículo, tanto en el modo manual como en el modo automático, se enviará al componente *Controlador.*
- **Controlador:** Componente de la aplicación que se ejecutará sobre un dispositivo en contacto directo con los actuadores del vehículo. Este componente, recogerá las peticiones de movimientos realizadas por el *Servidor* e interactuará con los actuadores para llevar a cabo los movimientos.

1.3. Objetivos

El objetivo general del proyecto es la creación de una aplicación que permita al usuario controlar un vehículo de forma remota o especificar una serie de movimientos y que el vehículo sea capaz de llevarlos a cabo de forma autónoma. Todo ello a través de un dispositivo móvil.

El objetivo general se divide en una serie de objetivos principales, los cuales se deberán de cumplir para que la aplicación pueda funcionar de forma correcta:

- **Desarrollo del componente Aplicación móvil:** Este componente, como su nombre indica, será una aplicación para dispositivos móviles, la cual permitirá al usuario elegir entre los distintos modos de control, y dependiendo del modo seleccionado ofrecer una interfaz para especificar los movimientos a realizar.
- **Desarrollo del componente Servidor*:*** Se encargará de recibir el modo de control elegido por él usuario mediante la *Aplicación móvil*, dependiendo de este modo de control, permitirá al usuario un control directo sobre el vehículo a través de la *Aplicación móvil*, o recogerá un conjunto de movimientos especificados por el usuario mediante la *Aplicación móvil y* los llevará a cabo cuando considere oportuno. La toma de decisiones en la conducción autónoma depende de la información recibida del entorno mediante imágenes capturadas por una cámara. Todo aquel movimiento que este componente desee que realice el vehículo se enviará al componente *Controlador* como una petición de movimiento.
- **Desarrollo del componente Controlador**: Se encargará de recibir las peticiones de movimientos del *Servidor* y procesarlas. Una vez procesadas, se enviarán a los distintos actuadores las señales eléctricas para llevar acabo el movimiento, los se encuentran conectados al dispositivo que ejecuta este componente. Por último, enviará una respuesta al *Servidor* dependiendo del resultado del movimiento.

Cada uno de estos objetivos principales, tiene además una serie de sub-objetivos que debe de cumplir, estos sub-objetivos son los siguientes:

- La interfaz ofrecida por la *Aplicación móvil*, tanto en modo automático como modo manual, debe ser intuitiva y fácil de utilizar.
- A través de la *Aplicación móvil* se deberá poder observar el entorno en el cual se encuentra el vehículo, tanto en el modo manual como en el modo automático.
- La *Aplicación móvil* debe permitir el almacenamiento de un trayecto para su posterior ejecución. Estos trayectos estarán compuestos de un conjunto de movimiento y se deberán de almacenar de forma persistente. Además, estos trayectos deben poder ser editados o borrados por el usuario.
- El *Servidor* se deberá de ejecutar sobre un hardware que soporte la conexión de una cámara, así como la toma, procesamiento y el envío de las imágenes tomadas por dicha cámara.

- El *Servidor* será el encargado de la toma, procesamiento y envío de las imágenes utilizadas por la aplicación, ya sea para mostrar el entorno del vehículo, como para la realización de la conducción autónoma.
- El *Servidor* solo permitirá una conexión de forma simultánea, esto permite tener la seguridad de que solo habrá un usuario controlando el vehículo.
- El *Controlador* deberá de ofrecer interfaz reutilizable para la realización de peticiones de movimiento, pudiendo ser usada para el control del vehículo tanto en el modo manual como en el modo automático.
- La aplicación debe de ser capaz de cambiar de modo automático a manual sin necesidad de reiniciar o cambiar ninguno de los componentes que la forman.
- Todos los componentes que forman parte de la aplicación deben ser tolerantes a errores, por lo tanto, no deben darse, cierres bruscos, ante cualquier error causado por el usuario.

1.4. Estructura del documento

En este apartado se ofrece un breve resumen de los distintos apartados que componen la documentación, conociendo así de antemano la información que se encuentra en cada uno de ellos:

1. **Introducción:** Intenta dar una visión global del proyecto, indicando las motivaciones que dieron origen al proyecto y los objetivos que se pretenden cumplir. Por último, se especifica cuál es la estructura que sigue el documento.
2. **Estado de la cuestión:** Analiza la situación actual, contemplando las distintas alternativas en cuanto a software y hardware necesario para llevar a cabo el proyecto. Al final de este apartado se realizará un breve resumen de las alternativas escogidas.
3. **Análisis:** Definición de los requisitos tanto de usuario como de sistema que debe cumplir la aplicación, así como las matrices de trazabilidad que indicaran la correspondencia entre ellos. Además, se realiza una recopilación de los distintos casos de uso de la aplicación.
4. **Diseño:** Descripción de la solución, indicando las distintas decisiones de diseño tomadas, tanto en la implementación de la aplicación como en el desarrollo de las distintas interfaces. En este apartado, también se indicarán las distintas librerías y funcionalidades que son utilizadas en la aplicación, pero han sido desarrolladas por terceros, indicando a su vez, el motivo de su selección.
5. **Marco regulador e impacto socioeconómico:** En este apartado, se realizará un con relación al marco regulador que afecta al desarrollo de este proyecto como vehículo autónomo. Después, observaremos la afectación que puede tener en el entorno socioeconómico.
6. **Planificación:** Indica cual ha sido la planificación que se ha llevado a lo largo de del desarrollo del proyecto.
7. **Presupuesto:** En este apartado, se especifica un presupuesto detallado respecto a los costes de realización del proyecto.
8. **Conclusiones y trabajos futuros:** Expone las conclusiones obtenidas una vez finalizado el proyecto, incluyendo posibles ampliaciones o mejoras que se puedan realizar, las cuales, no se ha podido llevar a cabo por falta de conocimiento o superar el presupuesto o el tiempo de desarrollo.
9. **Acrónimos, abreviaturas y definiciones:** En este apartado se indica el significado de los distintos acrónimos y abreviaturas utilizados a lo largo del documento, así como la definición de aquellas palabras que el lector pueda desconocer.
10. **Bibliografía:** Conjunto de las distintas referencias bibliográficas utilizadas para la realización del documento.

2. Estado de la cuestión

En este apartado se realizará un estudio de los coches controlados de forma remota y autónoma que existen actualmente y sus distintos usos. Después, se evaluarán las distintas alternativas, referentes a tecnologías, que encontramos a la hora de desarrollar la aplicación. Barriendo desde los distintos sistemas operativos móviles, con el objetivo de elegir aquel que se acomode más a las necesidades de la *Aplicación móvil*, hasta el hardware y lenguajes de programación utilizados para implementar y ejecutar los distintos componentes de la aplicación. Por último, se hará un breve resumen de las alternativas escogidas para la realización del proyecto.

2.1. Coches controlados de forma remota y autónoma

En este apartado se realiza un estudio respecto a los coches controlados de forma remota y autónoma, indicando los distintos usos que se le dan a este tipo de coches y los principales proyectos que existe en la actualidad para el desarrollo de este tipo de vehículos.

2.1.1. Coches controlados de forma remota

Actualmente los coches controlados remotamente no son nada del otro mundo, ya que llevan existiendo desde hace bastantes años. Estos siguen utilizándose desde el ocio hasta en investigación. Algunos ejemplos de uso de coches de conducción remota son los siguientes:

- **Ocio**

Un ejemplo de la utilización de los coches de conducción remota para el ocio, son los coches de radiocontrol. Estos son automóviles a escala reducida que se controlan mediante un dispositivo que envía señales de radio.

Ilustración 1 - Fotografía coche radio control

- **Profesionalmente**

En muchos documentales y películas se utilizan los coches de conducción remota para grabar aquellas escenas que una persona no puede grabar. Este es un ejemplo del uso de los coches de conducción remota en el ámbito profesional.

Ilustración 2 - Fotografía coche controlado remotamente para la grabación de documentales

- **Exploración e investigación**

Curiosity es una misión espacial organizada por la NASA para explorar la superficie de marte. Esta misión consiste en el envió de un *rover* a la superficie de marte y controlarlo desde la Tierra. Este rover, es un vehículo con aspecto muy parecido al de un coche, y está diseñado para ir sobre la superficie de astros. Este es un claro ejemplo del uso de los coches de coches de conducción remota para la exploración e investigación de lugares a los que una persona actualmente no puede ir.

Ilustración 3 - Fotografía del rover de la misión Curiosity de la NASA

2.1.2. Coches controlados de forma autónoma

En la actualidad, existen numerosas empresas tanto de sectores tecnológicos como del propio sector automovilístico que tienen en marcha proyectos para el desarrollo de vehículos autónomos.

El motivo de este afán por producir coches autónomos es debido a que abren una gran cantidad de nuevas posibilidades, sobre todo en el sector automovilístico y el de transporte de mercancías y personas. Ya que cuando se comercialice un modelo estable y que pueda llevar a cabo una conducción autónoma completa, no será necesario disponer de un conductor a los mandos del vehículo.

Con este tipo de vehículo, no solo cambiarán algunos sectores comerciales, sino que también cambiará la legislación vigente con respecto a la conducción de lo automóviles, ya que habrá gente que podrá utilizar estos vehículos sin tener conocimiento de cómo se conducen. Además, podrá llegarse a dar el caso de prohibir la conducción manual de este tipo de vehículos, si la conducción autónoma demuestra ser más segura que la manual.

A continuación, se hablará de algunos de los principales proyectos de vehículos de conducción autónoma que existen actualmente:

- **Tesla**

[B40] La empresa Tesla ha llevado a sus vehículos lo que ellos denominan *autopilot*, que es lo más parecido que hay actualmente a vehículos de conducción autónoma que puedan ser adquirido por cualquier persona. Este *autopilot* permite al vehículo autoconducirse, pero siempre debe estar presente un conductor, ya que esta conducción autónoma aún no está libre de fallos, la propia Tesla así lo indica. Los clientes que compren un vehículo podrán elegir entre 3 niveles de equipamiento para este *autopilot*:

- Nivel 1, básico: Dispone de una única cámara, permite el estacionamiento asistido y algunas funciones simple de navegación.
- Nivel 2, intermedio: Dispone de 4 cámaras, permite la conducción asistida con el *autopilot.*
- Nivel 3, máximo: Dispone de 8 cámaras, permite una conducción autónoma casi total.

Además, estos coches incorporan 12 sensores ultrasónico, un radar delantero y un ordenador de a bordo.

El funcionamiento del vehículo consiste en tomar la información dada por las cámaras, los sensores ultrasónicos y el radar delantero, y mediante el ordenador de a bordo, que utiliza una red neuronal desarrollada por Tesla, para procesar dicha información y realizar la toma de decisiones.

Ilustración 4 - Fotografía funcionamiento autopilot Tesla

- **Waymo**

[B41] Es la empresa de Google para el desarrollo de vehículos autónomos. Aún no ha comercializado ninguna unidad, pero ha producido 100 unidades autónomas sobre el automóvil Chrysler modelo Pacifica.

Los vehículos autónomos de esta empresa utilizan un dispositivo colocado en el techo del vehículo que se denomina LIDAR, el cual, mediante el uso de rayos de luz, mide la distancia a los objetos. Este dispone de 64 rayos láser y siempre está girando sobre si mismo, realizando 900 vueltas por minuto. El coche utiliza este dispositivo genera imágenes 3D del entorno que le rodea. Además, estos coches disponen de 4 radares, uno a cada lado del vehículo, un GPS, una unidad de medición de inercia y un ordenador de a bordo.

El funcionamiento del vehículo consiste en tomar la información por todos los sensores de los que dispone, procesar parte de esta mediante el ordenador de a bordo y el resto procesarla mediante el uso de los servidores en la nube.

Ilustración 5 - Fotografía coche autónomo Waymo

2.2. Estudio de las alternativas de sistema operativo móvil

En la actualidad existen una cantidad inmensa de dispositivos móviles y fabricantes, ofreciendo un sinfín de posibilidades a la hora de escoger dispositivo. Pero, a diferencia de esto, el número de sistemas operativos utilizados por estos dispositivos es mucho menor, siendo actualmente los más conocidos Android e iOS, excluyendo de esta lista a otros sistemas operativos como Symbian, Windows Phone o BlackBerry OS, por haber caído en desuso en los últimos años.

A continuación, se evaluarán las principales características de los principales sistemas operativos con el fin de escoger aquel que más se acomode a las necesidades del módulo del proyecto que corresponde a la *Aplicación móvil*. Por último, se realizará una breve comparativa y decisión sobre el sistema operativo móvil a utilizar.

2.2.1. Android

Ilustración 6 - Logotipo de Sistema Operativo Android

Android [B1] es un sistema operativo móvil desarrollado originalmente por Android Inc y comprado por Google en 2005, siendo este su actual propietario y principal desarrollador. Esta desarrollado sobre un núcleo de Linux, actualmente es el sistema operativo móvil con mayor cuota de mercado con diferencia y está completamente integrado con los principales servicios de Google. La última versión de este sistema operativo es la 8.0.0 Oreo, presentada el 21 de agosto de 2017. Este sistema operativo se utiliza principalmente en Smartphones y tablets.

Uno de los principales puntos fuertes de Android, es que dispone de una licencia de software GNU GPL, lo que permite que exista una gran cantidad de documentación, soporte, librerías, dispositivos que disponen de este sistema operativo, desarrolladores de aplicaciones y libertad a la hora de desarrollar.

Las aplicaciones están basadas en Java, además, el IDE (Android Studio) y SDK oficiales de esta plataforma, son directamente proporcionados y mantenidos por Google de forma gratuita, facilitando el desarrollo de aplicaciones.

El mercado oficial de aplicaciones de este SO móvil es Google Play. Los desarrolladores deben de pagar 25€ para darse de alta como tal y poder subir aplicaciones a este mercado oficial [B3]. Por otro lado, existen otra serie de mercados, como Amazon AppStore, los cuales son autorizados por Google, lo que demuestra que

sus aplicaciones son fiables, que permiten subir aplicaciones a su mercado de forma gratuita [B4].

Uno de los principales inconvenientes que se puede encontrar a la hora de desarrollar en Android, es la gran cantidad de dispositivos diferentes que disponen de este sistema, lo que obliga a tener especial atención a la hora de diseñar las interfaces de las aplicaciones.

Por otro lado, el soporte de actualizaciones del sistema operativo para los distintos dispositivos no es proporcionado por Google, sino que lo proporciona el fabricante del dispositivo, lo que provoca que muchos de los dispositivos se encuentren desactualizados, causando problemas de seguridad y la falta de funcionalidades en algunos dispositivos.

2.2.2. iOS

Ilustración 7 - Logotipo de Sistema Operativo iOS

iOS[B2] es un sistema operativo móvil desarrollado por Apple Inc, siendo este su único desarrollador. Esta desarrollado sobre un núcleo XNU y es actualmente el segundo sistema operativo móvil con mayor cuota de mercado. Se integra con los principales servicios de Apple y no puede ser instalado en hardware de terceros, pudiéndose solo instalar en dispositivos de la marca Apple. La última versión de este sistema operativo es la iOS 11, presentada el 5 de junio de 2017. Este sistema operativo es utilizado en los siguientes productos de la marca Apple: iPhone, iPod touch y iPad.

El principal punto fuerte de iOS es su optimización, permitiendo que el sistema operativo y las aplicaciones que funcionan sobre este se ejecuten de forma fluida. Esta optimización se debe a que el sistema operativo está especialmente diseñado para un número reducido de dispositivos.

Otra de las grandes ventajas es su soporte de actualizaciones, gracias a que este soporte es proporcionado directamente por Apple, lo que permite que la mayoría de los dispositivos se encuentren actualizados a la última versión del sistema operativo.

Cuenta con un lenguaje de programación propio denominado Swift [B5], desarrollado por Apple, especialmente enfocado para el desarrollo para iOS y macOS, lo que permite una mayor eficiencia en dispositivos con dichos sistemas operativos.

El IDE XCode y el SDK de iOS es ofrecido y mantenido por Apple de forma gratuita, pero es necesario de una licencia *Developer* de Apple para poder probar las aplicaciones en un dispositivo, esta licencia tiene un coste de 99€ [B6].

A este sistema operativo solo se le pueden instalar aplicaciones que se encuentran alojadas en el mercado de Apple denominado AppStore, siendo imposible instalar aplicaciones que no se encuentren en este mercado sin realizar modificaciones sobre el sistema operativo. Para poder publicar las aplicaciones en este mercado, es necesario disponer de la ya mencionada licencia de *Developer* de Apple.

Por último, la muy reducida variedad de dispositivos que utilizan este SO móvil facilita el diseño de las interfaces para las aplicaciones.

2.2.3. Comparativa y decisión

En este apartado se realizará la comparativa y toma de decisiones respecto sobre qué sistema operativo móvil se desarrollará la *Aplicación móvil*. Los sistemas operativos comparados serán los estudiados en los apartados anteriores, mostrando las principales ventajas en inconvenientes de los dos sistemas.

		
Nombre del SO	Android	iOS
Desarrollador	Google	Apple
Núcleo	Linux modificado	XNU
Lenguaje de programación del SO	Java, C y C++.	C, C++, Objetive-C y Swift.
Lenguaje de programación de las aplicaciones	Aplicaciones principalmente basadas en Java, pudiendo incluir código en C y C++.	Aplicaciones principalmente codificadas en Swift y Objetive-C.
Instalación de SO	Puede ser instalado en Hardware de terceros. Por lo que la cantidad y variedad de dispositivos es inmensa.	Solo puede ser instalado en dispositivos Apple. Reduciendo así la variedad de dispositivos que disponen del sistema operativo.
IDE y SDK	IDE Android Studio y SDK proporcionados por Google, sin limitaciones, pudiendo probar de forma gratuita las aplicaciones en dispositivos físicos.	IDE XCode y SDK proporcionados por Apple de forma gratuita, limitado si no se dispone de licencia Developer de Apple, siendo necesaria si se quiere probar las aplicaciones en dispositivos físicos.
Mercados	Mercado oficial Google Play, necesaria licencia de desarrollador de Google para subir aplicaciones. Varios mercados autorizados como Amazon AppStore, sin coste de para el desarrollador a la hora de subir aplicaciones.	Únicamente Apple Store, siendo necesario licencia Developer de Apple para subir aplicaciones.
Coste licencia desarrollador	25€	99€/año
Comunidad	Comunidad muy amplia debido a la licencia GNU GPL y el alcance del SO.	Comunidad menos numerosa y cerrada.
Soporte de actualizaciones	Responsabilidad de los fabricantes de dispositivos. Dispositivos más desactualizados, ocasionando falta de funcionalidades y problemas de seguridad.	Responsabilidad de Apple. Dispositivos más actualizados y con mayor tiempo de vida útil.

Tabla 1 - Comparativa aspectos principales Android e iOS

Observando las características de los dos sistemas operativos móviles, se decidió escoger **Android** para este proyecto por los siguientes:

- **Conocimiento previo de la plataforma:** Uno de los principales motivos por el cual se escoge Android, es el conocimiento previo que se tiene de la plataforma, ya que, a diferencia de en iOS, ya se han realizado proyectos en este sistema operativo. Conociendo, además, como se utiliza el IDE y SDK proporcionado por Google, y como se desarrolla el código y las interfaces gráficas para este tipo de aplicaciones.
- **Conocimiento de los lenguajes de programación de las aplicaciones:** Se conoce de antemano como se programa en los lenguajes de programación Java, C y C++. A diferencia de Swift y Objetive-C, los cuales no se han utilizado con anterioridad.
- **Mayor cantidad y variedad de dispositivo que disponen del sistema operativo**: Aunque inicialmente este proyecto no está pensado para llevarse al ámbito comercial, siendo puramente académico, si se decidiera llevar a este ámbito, interesa poder llevarlo a la mayor cantidad de dispositivos posible.
- **Pruebas en dispositivos físicos de forma gratuita:** Debido a que por el momento no es una aplicación pensada para el ámbito comercial, este es un gran punto a favor de Android, debido a que nos permite probar la aplicación en nuestro dispositivo físico sin necesidad de tener una licencia de desarrollador con su correspondiente coste.
- **Coste licencia desarrollador:** Si en un futuro interesará publicar la *Aplicación móvil* en el mercado oficial Google Play, el coste de la licencia para Android es un único pago de 25€, a diferencia de la ofrecida por Apple, que tiene un coste de 99€ anuales.
- **Comunidad muy amplia:** La comunidad que da soporte a este sistema operativo es muy extensa, ofreciendo, además, una gran cantidad de librerías, documentación y ejemplos.
- **Disponibilidad de diferentes dispositivos Android:** Se dispone de antemano de dispositivos Android en los cuales realizar las pruebas de la aplicación, cosa que no ocurre en el caso de iOS.

2.3. Estudio de las alternativas de hardware

En este apartado se estudiarán las distintas alternativas con respecto al hardware. Estas decisiones de hardware se dividirán en 3 apartados, cada uno corresponderá con el hardware encargado de ejecutar un componente del proyecto.

2.3.1. Alternativas hardware para aplicación móvil.

En este apartado se decidirá cuál es la alternativa con respecto a hardware que se encargará de ejecutar la *Aplicación móvil*. Debido a que el sistema operativo móvil escogido para la realización de este proyecto ha sido Android, por lo motivos antes citados, se reduce la decisión a solo aquellos dispositivos que dispongan de este sistema operativo.

Como ya se ha mencionado anteriormente, Android es el SO móvil con mayor cuota de mercado y por consiguiente mayor número de dispositivos diferentes. Esto permite un gran abanico de posibilidades, pero debido a que no se necesita un dispositivo con unas características específicas, más allá de que pueda conectarse a una red inalámbrica de área local mediante conexión WI-FI y tengan instalado como sistema operativo Android, la comparativa solo se realizará sobre los distintos dispositivos de los que ya se disponen y que cumplan dichas características.

2.3.1.1. Xiaomi Redmi Note 3 Pro Special Edition

Ilustración 8 - Fotografía Xiaomi Redmi Note 3 Pro Special Edition

Móvil desarrollado por la empresa Xiaomi y puesto en venta en Mazo de 2016. Tiene instalado el sistema operativo Android con la versión 6.0.1 Marshmallow. Dispone de una pantalla de 5.5 Pulgadas LCD IPS con una resolución 1080 x 1920 que ofrece una densidad de pixeles de 401ppi. Tiene un Procesador Snapdragon 650 MSM8956 con una arquitectura de 64 bits, una CPU de 6 núcleos, 2 de ellos ARM Cortex A72 con una velocidad de reloj de 1.8Ghz y los 4 siguiente ARM Cortex A53 con una velocidad de reloj a 1.4Ghz, y una GPU Adreno 510. Dispone una batería de 4050mAh y conectividad Wi-Fi, Bluetooth, 2G, 3G y 4G. Por último, el modelo del que se dispone tiene 2Gb de memoria RAM y 16GB de memoria interna [B7].

2.3.1.2. Xiomi Redmi 2

Ilustración 9 - Fotografía Xiaomi Redmi 2

Móvil desarrollado por la empresa Xiaomi y puesto en venta en enero de 2015. Tiene instalado el sistema operativo Android con la versión 4.4.4. Kitkat. Dispone de una pantalla de 4.7 Pulgadas LCD IPS con una resolución 720 x 1280 que ofrece una densidad de pixeles de 312ppi. Tiene un Procesador Snapdragon 410 MSM8916 con una arquitectura de 64 bits, una CPU de 4 núcleos, ARM Cortex A53 con una velocidad de reloj a 1.3Ghz, y una GPU Adreno 306. Dispone una batería de 2200mAh y conectividad Wi-Fi, Bluetooth, 2G, 3G y 4G. Por último, el modelo del que se dispone tiene 1Gb de memoria RAM y 8GB de memoria interna[B8].

2.3.1.3. Alcatel One Touch Pop C7 7041X

Ilustración 10 - Fotografía Alcatel One Touch Pop C7

Móvil desarrollado por la empresa Alcatel y puesto en venta en septiembre de 2013. Tiene instalado el sistema operativo Android con la versión 4.2 Jelly Bean. Dispone de una pantalla de 5 Pulgadas TFT con una resolución 480 x 854 que ofrece una densidad de pixeles de 196ppi. Tiene un Procesador MediaTek MT6582M con una arquitectura de 32 bits, una CPU de 4 núcleos, ARM Cortex A7 con una velocidad de reloj a 1.3Ghz, y una GPU Mali-400 MP2. Dispone una batería de 2000mAh y conectividad Wi-Fi, Bluetooth, 2G, 3G. Por último, el modelo del que se dispone tiene 1Gb de memoria RAM y 4GB de memoria interna [B9].

2.3.1.4. Vodafone Smart Speed 6

Ilustración 11 - Fotografía Vodafone Smart Speed 6

Móvil desarrollado por la empresa Alcatel para la empresa de telecomunicaciones Vodafone y puesto en venta en agosto de 2015. Tiene instalado el sistema operativo Android con la versión 5.0 Lolipop. Dispone de una pantalla de 4.5 Pulgadas LCD IPS con una resolución 480 x 854 que ofrece una densidad de pixeles de 218ppi. Tiene un Procesador Mediatek MT6735 con una arquitectura de 64 bits, una CPU de 4 núcleos, ARM Cortex A53 con una velocidad de reloj a 1.1Ghz, y una GPU Mali-T720. Dispone una batería de 1780mAh y conectividad Wi-Fi, Bluetooth, 2G, 3G y 4G. Por último, el modelo del que se dispone tiene 1Gb de memoria RAM y 8GB de memoria interna [B10].

2.3.1.5. Comparativa y decisión

A continuación, se comparan los aspectos principales de los distintos dispositivos de los que se dispone, eligiendo aquel que se adapte mejor a nuestras necesidades presentes y futuras.

				
Nombre del dispositivo	Xiaomi Redmi Note 3 Pro Special Edition	Xiaomi Redmi 2	Alcatel One Touch Pop C7 7041X	Vodafone Smart Speed 6
Fabricante	Xiaomi	Xioami	Alcatel	Alcatel
Fecha de salida	Marzo de 2016	Enero de 2015	Septiembre de 2013	Agosto de 2015
Versión Android	6.0.1. Marshmallow	4.4.4. Kitkat	4.1. Jelly Bean	5.0. Lolipop
Pantalla	5.5 “ LCD IPS 1080 x 1920, 401ppi	4.7” LCD IPS 720 x 1280, 312ppi	5” TFT 480 x 854, 196ppi	4,5” LCD IPS 480 x 854, 218ppi
Batería	4050mAh	2200mAh	2000mAh	1780mAh
Modelo procesador	Snapdragon 650 MSM8956	Snapdragon 410 MSM8916	MediaTek MT6582M	Mediatek MT6735
Arquitectura	64bits	64bits	32bits	64bits
CPU	6 núcleos, 2x ARM Cortex A72 a 1.8Ghz y 4x ARM Cortex A53 a 1.4Ghz	4 núcleos ARM Cortex A53 a 1.3Ghz	4 núcleos ARM Cortex A7 a 1.3Ghz	CPU 4 núcleos ARM Cortex A53 a 1.1Ghz
GPU	Adreno 510	Adreno 306	Mali-400 MP2	GPU Mali-T720
RAM	2GB	1GB	1GB	1GB
Memoria Interna	16GB	8GB	4GB	8GB
Conectividad	2G, 3G, 4G, Wi-Fi, Bluetooth	2G, 3G, 4G, Wi-Fi, Bluetooth	2G, 3G, 4G, Wi-Fi, Bluetooth	2G, 3G, 4G, Wi-Fi, Bluetooth

Tabla 2 - Tabla comparativa Hardware para aplicación móvil

A la vista de las características de los dispositivos disponibles, se selecciona el dispositivo **Xiaomi Redmi note 3 pro Special Edition**, por los siguientes motivos:

- **Versión de Android más actualizada:** El resto de dispositivos se encuentran muy desactualizados en lo que se refiere a sistema operativo, esto provoca que no dispongan de los últimos parches de seguridad y funcionalidades incluidas en versiones posteriores. Se escoge este dispositivo, ya que es el más

actualizado, y por lo tanto se minimiza el número de parches de seguridad y funcionalidades de las cuales no está provisto.

- **Procesador más potente:** Es el dispositivo con mayor potencia tanto en CPU y GPU, lo que permitirá a la *Aplicación móvil* ejecutarse de forma más fluida que en el resto de dispositivos. Además, si fuera necesario, este dispositivo soporta una mayor carga de trabajo sin que se produzcan bloqueos o retrasos.
- **Mayor cantidad de memoria RAM:** Es el dispositivo con mayor cantidad de memoria principal, lo que permitirá ejecutar aplicaciones con un mayor consumo de memoria en ejecución que en el resto de dispositivos.
- **Pantalla notablemente más grande y con mayor resolución:** Esto permite la creación de interfaces con un mayor número de elementos sin dar la sensación de saturación o siendo difícil su visión.

2.3.2. Alternativas hardware para Servidor

En este apartado se decidirá, de las distintas alternativas hardware, cuál será la encargada de ejecutar el componente de la aplicación que denominamos *Servidor*.

Desde un primer momento, se tuvo claro que este componente se debía ejecuta sobre un hardware que dispusiera de las siguientes características para cumplir los objetivos fijados:

1. **El hardware debe tener un tamaño reducido:** El hardware debe ser lo suficientemente pequeño y ligero para que pueda ser transportado en todo momento por el vehículo.
   Se quiere que el procesamiento de la aplicación se encuentre sobre el vehículo, siendo el dispositivo móvil encargado de ejecutar la aplicación, el único dispositivo externo, ya que es estrictamente necesario. Esto se debe a que la comunicación con dispositivos externos deberá de realizarse de forma inalámbrica, con los posibles retrasos que esto conlleva, debido a que no puede haber cables conectados al vehículo que dificulten su movilidad.
   Con esto se quedan excluidos los computadores de sobremesa y portátiles.
2. **Debe de disponer de soporte para conectarse a una red de área local inalámbrica:** Este hardware debe de incorporar una tarjeta de red Wi-Fi o dar la posibilidad de añadir una. Esto es necesario, ya que la comunicación entre la *Aplicación móvil* y el *Servidor* se realizará a través de una red de área local inalámbrica, y, además, como el hardware se encontrará constantemente sobre el vehículo, no podría estar conectado a la red mediante cables, ya que afectarían a la movilidad del vehículo.
3. **Se le debe poder conectar y gestionar una cámara:** Como ya se ha indicado anteriormente, uno de los objetivos del proyecto es que el usuario pueda observar el entorno del vehículo sin necesidad de encontrarse los dos en el mismo lugar.
   Por ello, el hardware debe de incorporar una cámara o dar la posibilidad de conectar una. Además, debido a que el componente *Servidor* será el encargado de la toma de imágenes, así como su transmisión y procesamiento,

el hardware debe de disponer de la suficiente potencia para poder realizar esto.

4. **Se le deben poder conectar actuadores:** Este hardware debe poder tener conectados actuadores y poder hacerlos funcionar, ya sea porque estén conectados directamente a él o a un dispositivo o modulo auxiliar. Como ya se ha comentado, el *Servidor* será el encargado de enviar peticiones al *Controlador,* encargado de interactuar con los actuadores. El *Controlador* deberá de ejecutarse en el mismo hardware que el *Servidor* o en un hardware que se encuentre sobre el vehículo conectado al hardware del *Servidor* de forma no inalámbrica, ya que se quiere reducir al máximo los posibles retrasos o perdidas.

Una vez observadas estas características, decidimos que la mejor opción era utiliza una SBC [B11] u ordenador de placa reducida.

Ilustración 12 - Ejemplo de SBC

Estas SBC son computadores de reducidas dimensiones que cuyos elementos están conectados mediante un único circuito. Estas placas están compuestas al menos de un microprocesador, una memoria RAM e interfaces de entrada y salida, es decir, todos los componentes esenciales de un computador. Estas placas ofrecen poca potencia de computo, pero también unas reducidas dimensiones y un bajo consumo energético.

A continuación, se muestran las características de las placas SBC más utilizadas para este tipo de proyectos, por ello, solo se evaluarán las placas desarrolladas por la fundación Raspberry Pi y BeagleBoard.org, debido a que son las placas con mayor número de proyectos similares, soporte, comunidad y documentación. Por último, se realizará una comparativa y decisión respecto a cuál se utilizará en el proyecto.

2.3.2.1. Raspberry Pi

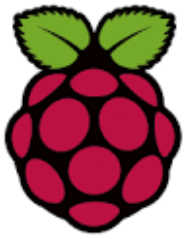

Ilustración 13 - Logotipo Raspberry Pi

Actualmente, Raspberry Pi [B12] es la placa SBC de bajo coste más conocida. Esta es desarrollada en Reino Unido por la fundación Raspberry Pi, esta fundación fue creada en 2009 como soporte a la enseñanza de las ciencias de la computación en las escuelas, promoviendo el aprendizaje de lenguajes de programación como Python, C o C++. Su sistema operativo oficial es una adaptación de la distribución de Linux Debian, llamada Raspbian, pero se da la posibilidad de instalar otros sistemas operativos desde versiones de Ubuntu a Windows 10.

Actualmente, todos los modelos de esta placa incorporan un SOC Broadcom a diferencia de las versiones iniciales no definitivas, que se basaban en un microcontrolador Atmel ATmega644. Además, incluyen una memoria RAM, una GPU, varios puertos USB, alimentación mediante puerto micro USB y un puerto de salida de video HDMI. En las primeras versiones A/B de la placa se disponía de 26 pines GPIO, aumentándose a 40 y añadiendo un puerto Ethernet en versiones posteriores. Por último, el almacenamiento no se incluye en estas placas, siendo necesario añadir una tarjeta SD en las versiones A, A+ y B del primer modelo, y en versiones posteriores, una tarjeta microSD.

Las distintas versiones que se han producido de esta placa son:

1. **Raspberry Pi Modelo A y A+:**

Ilustración 14 - Fotografía Raspberry Pi modelo A

El primer modelo desarrollador por la compañía fue la Raspberry Pi Modelo A, en el año 2012. Este modelo utilizaba un procesador Broadcom BCM2835[B18], una CPU ARM1176JZF-S con un único núcleo a 700Mhz, un procesador gráfico de doble núcleo VideoCore IV y disponía de 256MB de memoria RAM. Como ya se ha comentado, solo disponía de 26 pines GPIO y no disponía de puerto Ethernet, siendo necesario conectar

un adaptador de red mediante USB, debido a esto su único puerto USB quedaba ocupado. Por otro lado, también disponía de un conectar de cámara, un conector HDMI, uno de Video RCA y un puerto microUSB para la alimentación. Su coste inicial fue de 40€, encontrándose actualmente descatalogado.

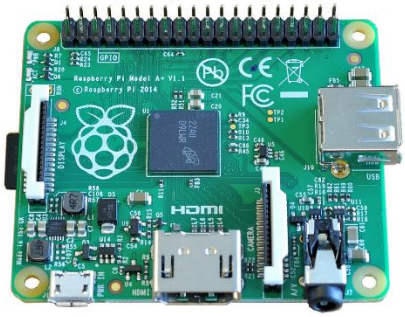

Ilustración 15 - Fotografía Raspberry PI modelo A+

Posteriormente se lanzó una actualización de este modelo, denominándose Raspberry Pi modelo A+. Este modelo eliminaba el puerto de Video RCA, aumentaba el número de pines GPIO a 40 y la memoria RAM a 512MB. Este modelo si sigue en venta por los distribuidores oficiales, vendiéndose en el distribuidor oficial más económico por un precio de 22,14€ [B12].

2. Raspberry Pi Modelo B y B+:

Estos modelos también fueron desarrollados en el año 2012, y son una variante de la Raspberry Pi Modelo A.

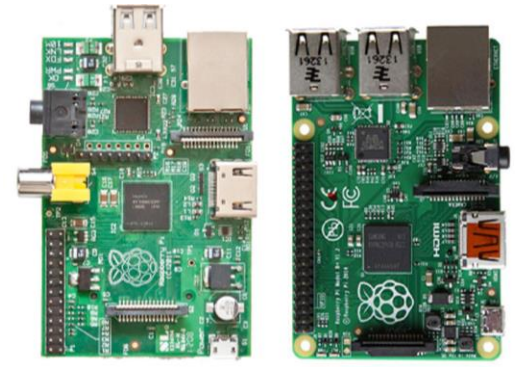

Ilustración 16 - Fotografía Raspberry PI B (Izquierda) y B+ (Derecha)

Para la Raspberry Pi Modelo B, al igual que la anterior A+, se aumentó su memoria RAM hasta los 512MB, pero se mantuvieron los 26 pines GPIO y el puerto de Video RCA. Por otro lado, se incluyó un segundo puerto USB y se introdujo el ansiado puerto de Ethernet. Este modelo tuvo un coste inicial de 40€, encontrándose actualmente descatalogado.

Posteriormente, al igual que para el modelo A, se realizó una actualización del modelo B, denominando a esta nueva tarjeta Raspberry Pi Modelo B+. Esta tarjeta incluía dos puertos USB más, sustituía el almacenamiento por tarjeta SD a tarjeta microSD, se aumentaron los pines GPIO a 40 y se volvía a eliminar el puerto de Video RCA. Al igual que ocurría con los modelos A/A+, el modelo B se encuentra

descatalogado, pero el modelo B+ puede encontrándose en el distribuidor oficial más económico por un precio de 29,03€ [B13].

3. Raspberry PI 2 Modelo B:

Ilustración 17 - Fotografía Raspberry Pi 2 modelo B

Modelo lanzado en el año 2014, mantiene todas las mejoras introducidas en el modelo Raspberry Pi Modelo B+. Este cambia el procesador utilizado anteriormente por un Broadcom BCM2836 [B19], que modifica la CPU ARM1176JZF-S de un solo núcleo a 700Mhz, a una CPU ARMv7 Cortes-A7 de 4 núcleos a 900Mhz, pero se mantiene el procesador gráfico VideoCore IV. Por último, aumenta la memoria RAM de 512MB a 1GB. Su coste actual es de en el 36,53€ en el distribuidor oficial más económico [B14].

4. Raspberry Pi Zero y Zero W:

Ilustración 18 - Fotografía Raspberry Pi Zero

La Raspberry Pi Zero es un modelo de tamaño reducido lanzado en 2015. Tiene el mismo procesador Broadcom BCM2835[B18] que las primeras versiones, pero en este caso, se consigue hacer funcionar la CPU a 1GHz, pero se mantienen el resto de características. Tiene una memoria RAM de 512MB, dispone de 2 puerto microUSB, uno para alimentación y otro como puerto de E/S, sustituye el puerto HDMI por un puerto miniHDMI, tiene un puerto CSI para cámara, mantiene el almacenamiento mediante microSD y dispone de 40 pines GPIO. Su coste es de 5,50€ con IVA, en el único distribuidor oficial que las comercializa [B15].

Ilustración 19 - Fotografía Raspberry Pi Zero W

Posteriormente, como ya sucedió con los modelos anteriores, se lanzó un modelo de actualización, La Raspberry Pi Zero W, a diferencia del anterior este incluye un módulo Wi-Fi y un módulo Bluetooth 4.1 Low Energy incorporados en la placa. Su coste es de 11€, en el único distribuidor oficial que las comercializa [B16].

5. **Raspberry Pi 3 Modelo B:**

Ilustración 20 - Fotografía Raspberry Pi 3 modelo B

Es el modelo más actual de la placa, fue lanzado en 2016, continua con la estela de las características de la Raspberry Pi 2 Modelo B, pero realiza algunos cambios. Modifica el de nuevo el procesador con un nuevo Broadcom BCM2837 [B20], que al igual que su predecesor continuo con una CPU de 4 núcleos, pero en este caso de tipo ARMv8 Cortex-A53 con 4 núcleos a 1.2GHz, pero se mantiene el procesador gráfico VideoCore IV. Por último, incluye un módulo Wi-Fi y un módulo Bluetooth 4.1 Low Energy incorporados en la placa. Su coste actual es de en el 36,53€ con IVA incluido, en el distribuidor oficial más económico [B17].

Antes de continua con la evaluación de las alternativas, se descartan de la comparativa final los modelos Raspberry Pi Modelo A/A+, B/B+, Zero y Zero W. Debido a que la potencia de la CPU ofrecida por estas placas es insuficiente para el proyecto que queremos llevar acabo. Quedando para la comparativa final los modelos Raspberry Pi 2 Modelo B y 3 Modelo B.

2.3.2.2. BeagleBoard

Ilustración 21 - Logotipo BeagleBoard

BeagleBoard [B21] es un SBC de hardware libre de bajo coste. Esta fue desarrollada por BeagleBoard.org, el propósito inicial al igual que en la Raspberry Pi fue educacional. No tiene un sistema operativo oficial, pero puede soportar numerosas distribuciones de Linux, incluso Symbian e Android. Las distintas versiones que se han producido de esta placa son:

1. **BeagleBoard rev.C4:**

Ilustración 22 - Fotografía BeagleBoard rev.c4

[B22] Primer modelo desarrollado, se comercializo en Julio de 2008 y se realizó su última revisión en mayo de 2009, la cual vamos a analizar. Dispone de un procesador OMAP3530, con una CPU ARM Cortex-A8 de un único núcleo a 720MHz, con un procesador gráfico PowerVR SGX530 y dispone de una memoria RAM de 256MB. Dispone de salidas de video HDMI y S-Video, una entrada y una salida de audio de tipo Jack, lector de tarjetas SD y MMC, un conector JTAG, un puerto serial RS-232, puerto de alimentación de tipo barril, un puerto Ethernet, un puerto microUSB y un puerto USB. Actualmente se encuentra descatalogada.

2. **BeagleBoard-xM :**

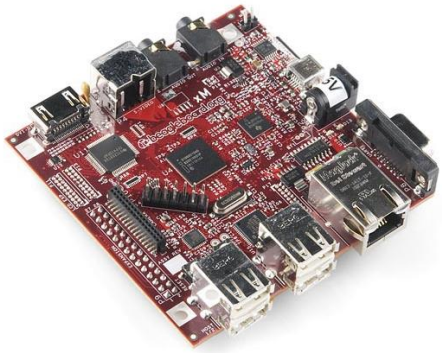

Ilustración 23 - Fotografía BeagleBoard XM

[B23] Actualización de la primera BeagleBoard, se comercializo en septiembre de 2010. Modifica el procesador introduciendo un DM3730, con una CPU ARM Cortex-A8 de un único núcleo a 1000MHz, mantiene el procesador gráfico y aumenta su memoria RAM hasta los 512MB. Además, incluye 3 puerto USB adicionales, un puerto para cámara y un puerto UART. Actualmente tiene un coste 136,75€, en el distribuido oficial más económico [B26].

3. **BeagleBone :**

Ilustración 24 - Fotografía BeagleBone

[B24] Modelo comercializado en octubre de 2011. Incorpora un procesador AM3359ZCZ72, con una CPU Cortex-A8 a 720MHz y un procesador gráfico PowerVR SGX530 y dispone de una memoria RAM de 256MB. Además, dispone de dos conectores de expansión de 46 pines cada uno, puerto Ethernet, dispone de alimentación mediante puerto miniUSB y de puerto de tipo barril, un puerto USB y un puerto JTAG. La principal característica de esta placa es que se pueden incluir una serie de *"capas de expansión"* que permite ampliar las características de la placa, permitiendo conectar desde una pantalla LCD táctil hasta una batería. Se pueden añadir un máximo de 4 capas simultaneas. Este modelo se encuentra descatalogado.

4. **BeagleBone Black :**

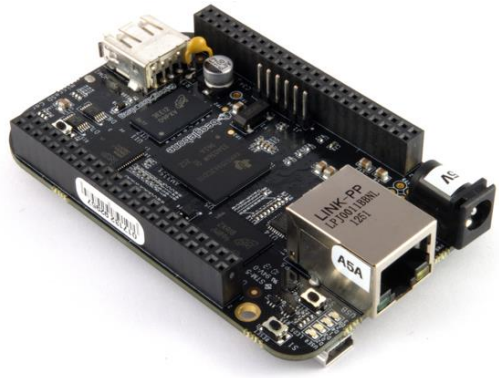

Ilustración 25 - Fotografía Beaglebone Black

[B25] Modelo comercializado en abril de 2013, es una actualización del modelo original BeagleBone. Incorpora un procesador AM3359ZCZ72, este se diferencia del procesador del modelo original únicamente en la frecuencia de reloj de la CPU, aumentándose a 1GHz. Aumenta la memoria RAM a 512MB e incluye una memoria Flash de 4Gb eMMC. Además, incluye un puerto miniHDMI y se mantienen el resto de características con respecto al modelo original. Actualmente tiene un coste de 55,33€, en el distribuidor oficial más económico [B27].

Una vez observados los distintos modelos ofrecidos por esta empresa, decidimos no incluir en la comparativa final ninguno de estos dispositivos. Estos dispositivos no proporcionan la suficiente potencia de CPU para la realización de este proyecto.

2.3.2.3. Comparativa y decisión

Una vez observadas las características de las placas SBC más utilizadas, solo nos quedamos con las placas **Raspberry Pi 2 Modelo B** y **Raspberry Pi 3 Modelo B**, ya que son las placas que cumplen en mayor medida nuestras necesidades.

		
Nombre del dispositivo	Raspberry Pi 2 Model B	Raspberry Pi 3 Modelo B
Desarrollador	Fundación Raspberry Pi	Fundación Raspberry Pi
Fecha de salida	2014	2016
SO	Raspbian, Ubuntu Mate, Ubuntu Snappy Core, Windows 10 IOT Core, OSMC, Libreelec, Pinet y Risc OS	Raspbian, Ubuntu Mate, Ubuntu Snappy Core, Windows 10 IOT Core, OSMC, Libreelec, Pinet y Risc OS
Modelo procesador	Broadcom BCM2836	Broadcom BCM2837
CPU	4 núcleos ARM Cortes-A7 a 900Mhz	4 núcleos ARMv8 a 1.2GHz
GPU	VideoCore IV	VideoCore IV
Memoria RAM	1GB	1GB
Puerto USB	4	4
Puerto CSI (Cámara)	1	1
Puerto HDMI	1	1
Salida audio Jack	1	1
Pines GPIO	40	40
Puerto Ethernet	1	1
Modulo Wi-Fi	Instalable	Integrado
Módulo Bluetooth	Instalable	Integrado
Alimentación	Puerto microUSB	Puerto microUSB
Almacenamiento	Tarjeta microSD	Tarjeta microSD
Precio	36,53€	36,53€

Tabla 3 - Comparativa Raspberry Pi 2 Modelo B y Raspberry pi 3 Modelo B

Observando las características se decidió utilizar para este proyecto la placa SBC **Raspberry Pi 3 Modelo B con SO Raspbian**, por los siguientes motivos:

- **Procesador:** El procesador del modelo escogido es algo superior, lo que nos asegura mejores tiempos de respuesta.

- **Inclusión de módulo Wi-Fi:** El modelo escogido dispone de un módulo Wi-Fi integrado, lo que nos ahorra la necesidad de comprar uno, con el sobrecoste que esto conlleva.
- **Coste:** La diferencia de coste entre los dos modelos es nula, ofreciendo la Raspberry Pi 3 Modelo B varias características bastante útiles para nuestro proyecto por el mismo precio.

Otros puntos a favor que hicieron decantar la balanza a las placas de la fundación Raspberry son los siguientes:

- **Permiten la instalación de una cámara:** Los dos modelos permite instalar una cámara, ya sea mediante el Puerto CSI como por un puerto USB.
- **SO instalables:** Permiten una gran variedad de sistemas operativos, pudiendo escoger aquel que mejor se adapte a nuestras necesidades. En este caso, utilizaremos **Raspbian** ya que es el SO oficial y se tiene experiencia en distribuciones de Linux.
- **Pines GPIO:** Permiten la conexión directa de actuadores a la placa. Además, mediante estos pines se pueden incorporar de *Raspberry Pi HATs,* que permiten ampliar las características de la placa.

2.3.3. Alternativas hardware para Controlador

En este apartado se decidirá, de las distintas alternativas hardware, cuál será la encargada de ejecutar el componente de la aplicación denominado *Controlador*.

El componente *Controlador* se encarga de realizar las peticiones de movimiento del componente *Servidor*, por lo tanto, se debe ejecutar en un hardware que interactúe de forma directa con los actuadores del vehículo. Este hardware, al menos necesita tantas conexiones como actuadores del vehículo se utilicen para realizar los movimientos, además, estas conexiones deben suplir las necesidades de corriente y tensión de los actuadores.

Los actuadores de los que dispone el vehículo que vamos a utilizar son los siguientes:

	Unidades	Voltaje	Amperaje	Descripción
	3	3 - 7,2V	min 60mA	Motor DC
	2	3 - 6V	-	Servomotor SG90 Tower Pro

Tabla 4 - Lista de actuadores vehículo

Además, uno de los motores DC, el encargado de la dirección del vehículo, tiene conectado un potenciómetro, el cual varia su valor dependiendo del giro del motor. Por lo tanto, también se necesita que el hardware disponga un pin de 5V de

alimentación, otro de tierra y una entrada analógica o digital con la que poder leer el valor del potenciómetro.

A la vista de las características de los actuadores, se descarta la posibilidad de utilizar el mismo hardware para este componente que para el componente *Servidor*. Ya que los pines GPIO, de la Raspberry Pi 3 Modelo B, solo ofrecen un corriente de 50mA, siendo insuficiente a la hora de utilizar los motores DC.

A continuación, se indicarán las características de las distintas soluciones barajadas para solucionar este problema. Por último, se realizará una comparativa de estas soluciones, escogiendo aquella que se adapte a las necesidades del proyecto.

2.3.3.1. Raspberry Pi HAT

Como ya se mencionaron en la comparativa del hardware del componente *Servidor*, Las placas Raspberry ofrecen la posibilidad de añadir a sus placas unos módulos de expansión conectados a los puertos GPIO, los cuales permiten aumentar las capacidades de la placa. Estos módulos son denominados Raspberry Pi HATs.

Existe una gran variedad de estos módulos, pero en este caso, el módulo que mejor suple nuestras necesidades es el siguiente:

- **Adafruit DC & Stepper Motor HAT:**

Ilustración 26 - Fotografía Adafruit DC & Stepper Motor HAT

[B30] Raspberry Pi HAT desarrollado por Adafruit. Dispone de 4 puentes H, que gracias al chipset TB6612, permite controlar 4 motores DC bidireccionales, con una corriente máxima de 1,2A y un voltaje comprendido entre 4,5 – 13,5V. Este chipset también permite controlar un máximo de 2 motores paso a paso unipolares o bipolares. Permite una alimentación externa de 5 - 12 V, además, incluye un jumper que permite separar la alimentación del Raspberry Pi HAT de la placa a la que se encuentra conectada. Por último, dispone de numerosas entradas disponibles y librerías para Python para su utilización. Coste del producto 22,50€.

2.3.3.2. Utilización de un microcontrolador Arduino UNO + Shield

La segunda solución propuesta, es conectar mediante USB un microcontrolador Arduino UNO a la Raspberry, utilizando el puerto serial del dispositivo Arduino como mecanismo de comunicación entre ambos. Pero antes de continuar, hablaremos un poco de la empresa y los dispositivos que oferta.

Ilustración 27 - Logotipo de Arduino

Arduino es una plataforma de hardware y software libre. Al igual que ocurría en Raspberry y BeagleBoard, el propósito de esta plataforma es acercar la electrónica y la programación a aquellos que no tienen conocimientos técnicos de estos temas o están comenzando a adquirirlos. Estas placas, permiten realizar mediante electrónica sencilla y programación un gran abanico de posibles proyectos multidisciplinarios.

Gracias a que es una plataforma intuitiva y fácil de usar, es utilizada ampliamente en proyectos de electrónica, tanto en el ámbito educacional como en el diseño de prototipos en el ámbito profesional, lo que conlleva que disponga de una gran cantidad de documentación, ejemplos, proyectos, soporte y una amplia comunidad.

Las principales características físicas de estas placas son las siguientes:

- **Microcontrolador**: Todas las placas Arduino disponen de un microcontrolador, este es elemento central de la placa, encargado de ejecutar las instrucciones grabadas en su memoria. Los más utilizados son el ATMega168, ATMega328 y ATMega1280.
- **Pines E/S**: Estas placas disponen de pines de entrada/salida digitales y pines de entrada analógica, el número total de pines varía dependiendo de la versión de la placa. Estos pines trabajan a una tensión de 5V una corriente máxima de 40mA. Algunos de los pines soportan PWM.
- **Puerto Serial**: Todos los dispositivos disponen de un puerto serial mediante él se conecta al PC y se carga el programa que deben ejecutar, este puerto puede disponer de distintos formatos, pudiendo ser USB tipo B, microUSB, etc.

El lenguaje de programación utilizado para programar los *sketchs* de estas placas es C, sin embrago, las librerías se crean a partir de clases programadas en C++. Estas placas también se pueden programar mediante ensamblador Atmel.

El IDE oficial de la plataforma es proporcionado por Arduino de forma gratuita con versiones en Linux, Mac y Windows. Este IDE proporciona todo lo necesario para desarrollar programas para estas placas, permitiendo crear fichero de programa,

compilarlos y cargarlos a la placa. El IDE también da la posibilidad de utilizar librerías desarrolladas por Arduino que ofrecen funcionalidades ya implementadas como puede ser la Liberia *Serial*, que permite la comunicación por el puerto serie, o la librería Servo, que permite el control de servo motores de forma sencilla, además, también permite la inclusión de librerías de terceros o el desarrollo de librerías propias.

Gracias a que estas placas transmiten y reciben información median el puerto serial, se puede pueden comunicar a la mayoría de lenguajes de programación populares como: Java, Python, Ruby, Matlab, etc. Ya sea porque soporten este tipo de comunicación de forma nativa, o mediante el uso de librerías de terceros.

Al igual que Raspberry, Arduino ofrece la posibilidad de añadir extensiones que le añadan características de las cuales no dispone, en este caso son denominados escudos o *shields*. Estos suelen ser desarrollados por terceros, proporcionando en muchos casos librerías que permitan el uso de estos *shields.*

Existen una gran variedad de placas de esta marca, pero debido a que no se necesitan unas características específicas, ya que solo se necesita poder gobernar unos pocos actuadores y queremos reducir al máximo el coste de nuestro proyecto, decidimos utilizar una placa Arduino UNO Rev3 de la cual disponemos. las características principales de esta placa son las siguientes:

- **Arduino UNO rev3:**

Ilustración 28 - Fotografía Arduino UNO rev3

[B32] Es la última revisión de esta versión de la placa. Utiliza un microcontrolador ATmega328 [B34], que funciona a una frecuencia de reloj de 16MHz, una memoria flash de 32KB, 1KB de memoria EEPROM, 2KB de SRAM, 23 líneas de E/S de propósito general, 32 registro de propósito general, 3 contadores de ticks de reloj y opera entre 1.8 - 5.5V. El puerto de conexión al PC es un USB tipo B manejado por un Atmega16U2. Este dispone de 6 pines de entrada analogía y 11 pines de entrada/salidas digitales, teniendo la posibilidad de realizar en 6 de ellos PWM.

Al igual que ocurría con la placa Raspberry Pi 3 Modelo B, los pines de estas placas están limitados a 5V de tensión y 40mA de corriente. Por ello, es necesario añadir los ya mencionados *Shields* de Arduino para cumplir con las exigencias de los actuadores. Los *Shields* que mejor suple las necesidades del proyecto son los siguientes:

- **Adafruit Motor/Stepper/Servo Shield v1.2**

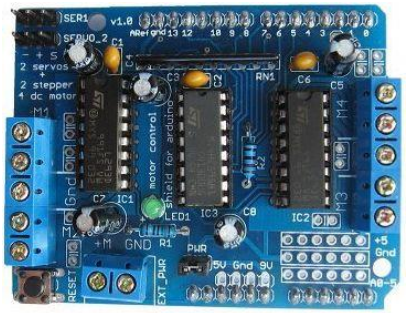

Ilustración 29 - Fotografía Adafruit Motor/Stepper/Servo Shield v1.2

[B33] *Shield* desarrollado por Adafruit. Dispone de 4 puentes H, que gracias al chipset L293D, permite controlar 4 motores DC bidireccionales, con una corriente máxima de 0,6A con picos de 1,2A y un voltaje comprendido entre 4,5 – 25V. Este chipset también permite controlar un máximo de 2 motores paso a paso unipolares o bipolares. Dispone de 2 conexiones para servomotores de 5V. Permite una alimentación externa de 5 - 25V, además, incluye un jumper que permite separar la alimentación del *Shield* y el Arduino.

Este *Shield* es compatible con los Arduino Mega 1280 y 2560, Diecimila, Duemilanove y UNO. Adafruit proporciona una librería para el IDE oficial de Arduino, que permite de forma simple utilizar los motores, pudiendo especificar valores de velocidad entre 1 y 255. Por último, cabe destacar que solo quedan libres los pines de entrada analógicos, de 5V y conexiones a tierra. El producto oficial está actualmente descatalogado, pero debido a que Adafruit es una Open-Source Hardware [B28], se pueden encontrar versiones de no originales de este modelo, con las mismas características y compatibles con las librerías de Adafruit, por 4€ [B35].

- **Adafruit Motor/Stepper/Servo Shield v2**

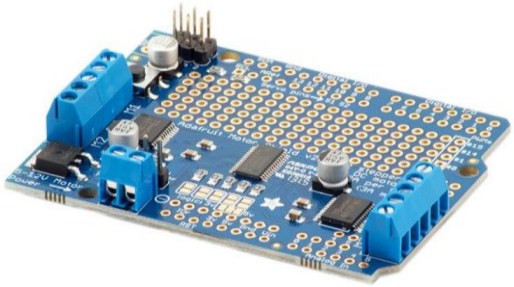

Ilustración 30 - Fotografía Adafruit Motor/Stepper/Servo Shield v1.2

[B34] Actualización de Adafruit al recién mencionado v1.2, la principal diferencia con el anterior es el cambio del chipset L293D de los puentes H por un chipset TB6612 que puede proveer de una tensión de 1.2A con picos de 3A y un voltaje de 4,5 -13,5V. En esta versión no se oculta el botón de reset del Arduino. Por último, este *Shield*, a diferencia del anterior, deja libres la mayoría de los pines del Arduino al que se encuentra conectado. El producto oficial cuesta 24,80€.

2.3.3.3. Comparativa y decisión

Una vez observadas las características de los posibles hardware que ejecutarán el componente *Controlador*, se realiza una comparativa de dichas características y se toma una decisión respecto a cuál utilizar para el proyecto:

		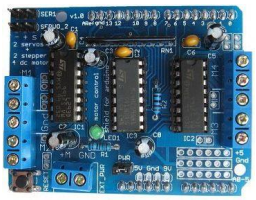	
Nombre	Adafruit DC & Stepper Motor HAT	Adafruit Motor/Stepper/Servo Shield v1.2	Adafruit Motor/Stepper/Servo Shield v2
Hardware adicional	No	Arduino UNO	Arduino UNO
Librerías	Disponible para Python	Disponible para Arduino IDE	Disponible para Arduino IDE
Motores DC controlables	4	4	4
V Motores DC	4,5 - 13,5V	4,5 - 25V	4,5 - 13,5V
A Motores DC	1,2A	0,6A con picos de 1,2A	1,2A con picos de 3A
Motores paso a paso controlables	2	2	2
Conexiones digitales libres	Si	No	Si
Conexiones analógicas libres	Si	Si	Si
Conexiones servomotores	No	2 a 5V	2 a 5V
Alimentación externa separada	Si	Si	Si
Coste	22,50€	4€	24,80€

Tabla 5 - Comparativa Adafruit DC & Stepper Motor HAT, Adafruit Motor/Stepper/Servo Shield v1.2 y v2

Observando las características de las distintas alternativas, se decidió utilizar para este proyecto la alternativa **Arduino UNO + Adafruit Motor/Stepper/Servo Shield v1.2**, por los siguientes motivos:

- **Cumple las necesidades de tensión y corriente:** Ofrece la posibilidad de controlar 4 motores DC de 4,5 - 25V y con una corriente máxima de 0.6A, suficiente para la alimentación de los 3 motores DC utilizados.
- **Dispone de conexiones para servomotores:** Ofrece dos conexiones para servomotores de 5V, justo las necesarias para este proyecto, a diferencia de la alternativa Adafruit DC & Stepper Motor HAT que no dispone de estas conexiones dedicadas.
- **Entorno de programación:** Formado por Arduino IDE y con lenguaje de programación C/C++, los cuales se conocen y han sido utilizados con anterioridad. Además, las librerías ofrecidas por Adafruit para el uso de este *shield* están diseñadas para ser incluidas y utilizadas mediante IDE oficial de Arduino. A diferencia de como sucede en la alternativa Adafruit DC & Stepper Motor HAT, cuyas librerías están diseñadas para ser usadas en Python, lenguaje del que se tiene menor conocimiento y experiencia.
- **Pines de entradas analógicas, 5V y a tierra:** Estas entradas libres permiten conectar y obtener el valor del potenciómetro antes mencionado.
- **Coste reducido:** El coste de esta alternativa es muy inferior al del resto, ofreciendo características muy similares. Esto se debe a que es un componente Open-Source Hardware con varios años de antigüedad, lo que permite encontrar versiones no originales que ofrecen las mismas características. Este punto es el que hace que se decante la balanza hacia la alternativa escogida, ya que hasta el momento dicha alternativa y la alternativa Adafruit Motor/Stepper/Servo Shield v2 cumplían los puntos anteriormente citados.

2.4. Estudio de las alternativas de Lenguaje de Programación.

En este apartado se estudiarán las distintas alternativas con respecto al lenguaje de programación a utilizar en los distintos componentes que conforman la aplicación. El apartado se dividirá en 3 subapartados, en los cuales se decidirá cuál es el lenguaje de programación a utilizar en cada uno de los componentes dependiendo de las necesidades de los mismos.

2.4.1. Alternativas de lenguaje de programación para aplicación móvil

En este apartado se evaluará y decidirá cuál es el lenguaje de programación utilizado para la *Aplicación móvil*.

Existe una gran variedad de lenguajes de programación a la hora de desarrollar aplicaciones móviles, pero en este caso, como el sistema operativo móvil escogido parar ejecutar la *Aplicación móvil* ha sido Android, las alternativas se reducen.

Además, como para este proyecto se quiere utilizar el IDE y SDK oficial de la plataforma, ya que disponen de una amplia documentación, comunidad, ejemplos y soporte. Este IDE y SDK, están diseñados para programar en Java con la posibilidad de incluir código en C y C++, por lo tanto, las alternativas quedan reducidas a estos lenguajes.

Esto nos lleva entonces a que el lenguaje a utilizar para programar la *Aplicación móvil* sea **Java con C/C++.**

2.4.2. Alternativas de lenguaje de programación para Servidor

En este apartado se evaluará y decidirá cuál es el lenguaje de programación utilizado para implementar el componente *Servidor*.

Este componente, a diferencia del resto de componentes del proyecto, puede ser programado en una infinidad de lenguajes de programación, ya que el hardware sobre el que se ejecuta es una placa Raspberry Pi 3 Modelo B con SO Raspbian, es decir, un computador con procesador ARM y una distribución de Linux como SO.

Debido a esta gran variedad de lenguajes, nos centraremos en escoger entre los más utilizados para programar sobre esta placa, ya que serán los que dispongan de un mayor soporte, comunidad y ejemplos. A continuación, se mostrarán las características de cada uno de ellos, y al final, se realizará una comparativa de sus características, escogiendo aquel que más se adapte a las necesidades del proyecto.

2.4.2.1. Python

Ilustración 31 - Logotipo Python

[B36] Lenguaje de programación publicado en 1991, cuyo desarrollador actual es *Python Software Foundation*. Es un lenguaje de programación interpretado, de tipado dinámico y que soporta los paradigmas de programación orientada a objetos, programación imperativa y programación funcional.

Es un lenguaje de alto nivel, que permite despreocuparse del manejo de memoria del programa, dispone de una sintaxis sencilla y fácil de aprender y dispone de una gran cantidad de librerías para esta placa.

2.4.2.2. C++

Ilustración 32 - Logotipo C++

[B37] Lenguaje de programación publicado en 1983, cuyo desarrollador principal fue Bjarne Stroustrup. Fue creado con la idea de extender el lenguaje de programación C, añadiendo la programación orientada a objetos. Es un lenguaje de programación compilado, de tipado estático y que soporta los paradigmas de programación orientada a objetos, programación genérica y programación estructurada.

Es un lenguaje de programación de nivel medio, con bastantes características de los lenguajes de programación de bajo nivel, lo que obliga a una mayor gestión de la memoria por parte del programador, con las ventajas y problemas que esto conlleva, además, debido a su cercanía al hardware permite una gran eficiencia en sus problemas. Por último, cabe destacar que tiene una sintaxis complicada, tiene mayor dificultad de aprendizaje que otros lenguajes y dispone también de una gran cantidad de librerías para esta placa.

2.4.2.3. Comparativa y decisión

A continuación, se realizará la comparativa entre los lenguajes de programación evaluados, tomando una decisión con respecto a cuál de ellos se va utilizar para implementar el componente *Servidor*.

		
Nombre	Python	C++
Año publicación	1991	1983
Tipo de lenguaje	Interpretado	Compilado
Tipado	Dinámico	estático
Paradigmas soportados	Programación orientada a objetos, programación imperativa y programación funcional.	Programación orientada a objetos, programación genérica y programación estructurada.
Nivel de abstracción	Alto nivel	Medio nivel
Ventajas	Facilidad de uso. Sintaxis sencilla y fácil de aprender No necesita compiladores. Dispone de un gran número de librerías para utilización de características de la placa.	Buena optimización, debido a cercanía al hardware. Mayor gestión de la memoria por parte del desarrollador. Dispone de un gran número de librerías para utilización de características de la placa. Dispone de todas las funcionalidades de C.
Inconvenientes	Optimización de los programas complicada.	Difícil de utilizar en comparación con otros lenguajes, debido a su sintaxis complicada y cerrada. Necesidad de utilizar compiladores.

Tabla 6 - Comparativa Python y C++

Observando las características de las dos alternativas, se decidió utilizar **C++** para la implementación de este componente. Los motivos de esta decisión son los siguientes:

- **Optimización:** Este lenguaje es a medio nivel, lo que permite una mayor cercanía al hardware y unas mayores posibilidades de optimización. En este caso, al ejecutarse el componente sobre un hardware cuya potencia es limitada, tener la posibilidad de optimizar al máximo el programa es una gran ventaja.
- **Dispone de librerías:** Este lenguaje, al igual que la alternativa no escogida, dispone de un gran número de librerías para explotar las características de esta placa.

- **Dispone de programación orientada a objetos:** Este lenguaje soporta el paradigma de programación orientada a objetos, a diferencia de otros lenguajes de nivel medio como puede ser C.
- **Lenguaje de programación conocido:** Este es un lenguaje que ya se ha utilizado en otros proyectos, por lo tanto, el principal inconveniente de este lenguaje, la dificultad de aprendizaje se minimiza en gran medida. Además, dispone de todas las funcionalidades de C, lenguaje que también es conocido y utilizado con anterioridad. Python a diferencia de C++, su aprendizaje es sencillo, pero no se ha utilizado con anterioridad, por lo tanto, no se tiene experiencia en él.

2.4.3. Alternativas de lenguaje de programación para Controlador

En este apartado se evaluará y decidirá cuál es el lenguaje de programación utilizado para implementar el componente *Controlador*.

En este caso, al igual que ocurría a la hora de elegir el lenguaje de programación del componente *Aplicación móvil*, no existen diversas alternativas. Esto se debe a que el hardware donde se va a ejecutar este componente es un Arduino UNO el cual solo puede ser programado mediante C/C++ y ensamblador Atmel, pero como a este se le incluye el *shield* Adafruit Motor/Stepper/Servo Shield v1.2, cuyas librerías para su uso están diseñadas para ser usadas mediante el IDE de Arduino, se reducen las alternativas únicamente a C/C++, ya que este IDE está diseñado para crear programas para la plataforma Arduino en dichos lenguajes.

Por lo tanto, el lenguaje de programación utilizado para implementar el componente *Controlador* es **C/C++.**

2.5. Resumen de las tecnologías escogidas

En este apartado se realiza un breve resumen de las tecnologías escogidas para la realización del proyecto, con el fin de agruparlas en un único apartado. Al igual que se hizo al realizar la toma de decisiones con respecto al hardware y los distintos lenguajes de programación, se dividirá este resumen en tres apartados, uno por cada componente que conforma la aplicación.

2.5.1. Alternativas tecnológicas escogidas para Aplicación móvil

Las alternativas escogidas para implementar y ejecutar el componente *Aplicación móvil* son las siguientes:

- **Hardware:** Xiaomi Redmi Note 3 Pro Special Edition.
- **Sistema operativo móvil:** Android.
- **Lenguaje de programación:** Java con C/C++.

2.5.2. Alternativas tecnológicas escogidas para Servidor

Las alternativas escogidas para implementar y ejecutar el componente *Servidor* son las siguientes:

- **Hardware:** Raspberry Pi 3 Modelo B.
- **Sistema operativo:** Raspbian (Distribución de Linux).
- **Lenguaje de programación:** C++.

2.5.3. Alternativas tecnológicas escogidas para Controlador

Las alternativas escogidas para implementar y ejecutar el componente *Controlador* son las siguientes:

- **Hardware:** Arduino UNO + Adafruit Motor/Stepper/Servo Shield v1.2.
- **Lenguaje de programación:** C/C++.

3. Análisis

En este apartado se realizará el análisis de sistema, obteniendo así los requisitos de usuario, casos de uso y requisitos de sistema. Además, se incluirán matrices de trazabilidad que indicarán la correspondencia entre requisitos de usuario y requisitos de software.

3.1. Requisitos de usuario

En este apartado se definirán los requisitos de usuario de forma detallada, completa y no ambigua. Los tipos de requisitos de usuario que se pueden encontrar son:

- **Requisitos de capacidad:** Definen aquellas funcionalidades y operaciones necesarias para los usuarios para resolver un problema o cumplir un objetivo.
- **Requisitos de restricción:** Definen las restricciones impuestas por los usuarios respecto a cómo se debe resolver un problema o como cumplir un objetivo.

Para obtener un mayor orden y ayudar a la compresión de los requisitos, estas dos grandes divisiones se dividen en subtipos:

- **Requisitos funcionales**: Hacen referencia a aquellas operaciones, problemas o funcionalidades que el sistema debe de cumplir.
- **Requisitos de rendimiento:** Hacen referencia a aquellos aspectos respecto a utilización de recursos que el sistema debe de cumplir, como pueden ser tiempos de respuesta, utilización de memoria principal, etc.
- **Requisitos de seguridad:** Definen una serie de características que el sistema debe de cumplir para mantener la seguridad del sistema**.**
- **Requisitos de implantación:** Hacen referencia a las distintas infraestructuras necesaria en el entorno en el que va a funcionar el sistema.

Los requisitos se recogerán en tablas las cuales tendrán el siguiente formato:

ID			
Título			
Descripción			
Prioridad		**Verificabilidad**	
Necesidad		**Estabilidad**	

Tabla 7 - Ejemplo formato Requisito de Usuario

- **ID:** Identificador univoco del requisito, cada requisito tendrá su propio ID que lo identificará con respecto al resto de requisitos. Este tendrá el siguiente formato:

RU-X-YZZ

- *RU:* Indica que es un requisito de usuario.
- *X:* Indica de que tipo es el requisito, tomará los siguientes valores:
  - *C:* Si el requisito es de capacidad.
  - *R:* Si el requisito es de restricción.
- *Y:* Indica en a que subcategoría pertenece el requisito, pudiendo tomar los siguientes valores:
  - *F:* Si el requisito pertenece a la subcategoría de requisitos funcionales.
  - *P:* Si el requisito pertenece a la subcategoría de requisitos de rendimiento.
  - *S:* Si el requisito pertenece a la subcategoría de requisitos de seguridad.
  - *I:* Si el requisito pertenece a la subcategoría de requisito de implantación.
- *ZZ:* Indica el número de requisito, la asignación de este número dependerá de su orden de creación, categoría y subtipo.

- **Título:** Palabra o conjunto de palabras que resume el contenido del requisito, sin entrar en detalles del contenido del mismo.
- **Descripción:** Explicación detallada de aquello de lo que trata el requisito.
- **Prioridad:** Indica la prioridad que tiene el requisito dentro del proyecto. Los valores que puede tomar este campo son:
  - *Alta***:** Indica el mayor nivel de prioridad, asignándose a aquellos requisitos que se deben de cumplir con mayor celeridad.
  - *Media:* Indica el nivel de prioridad intermedio, se asignará a aquello requisitos que, aun siendo necesarios para el funcionamiento del sistema, no lo son lo suficiente para priorizar su realización.
  - *Baja:* Indica el menor nivel de prioridad, se asignará a aquellos requisitos que puedan ser retrasados.
- **Estabilidad***:* Indica la estabilidad del requisito a través del tiempo, es decir, si el requisito se mantendrá constante durante todo el proyecto o podrá ser modificado. Los valores que puede tomar este campo son:
  - *Si:* Indicará que el requisito es estable, manteniéndose a lo largo de todo el proyecto.
  - *No:* Indicará que el requisito no es estable, pudiéndose dar cambios en el mismo.
- **Necesidad**: Indica como de necesario es un requisito para el proyecto. Los valores que puede tomar este campo son:
  - *Esencial*: Indicará que el requisito es imprescindible para el proyecto, imposibilitando que este pueda ser descartado en un futuro.
  - *Desechable*: Indicará un requisito que no es imprescindible para el proyecto, pudiendo ser descartado en etapas posteriores del desarrollo.

- **Verificabilidad:** Indica el nivel de dificultad que tiene un requisito para ser verificado. Los valores que puede tomar este campo son:
    - *Alta:* Indica el máximo nivel de verificabilidad, indicando que el requisito puede ser verificado de forma sencilla.
    - *Media:* Indica el nivel medio de verificabilidad, indicando que se pueden encontrar algunos problemas a la hora de verificar el requisito.
    - *Baja:* Indica el nivel más bajo de verificabilidad, indicando que verificar el requisito puede ser una tarea complicada de llevar acabo.

3.1.1. Requisitos de capacidad

En este aparatado se especificarán los distintos requisitos de usuario de capacidad que el sistema debe de cumplir.

3.1.1.1. Requisitos funcionales

En este apartado se muestran los requisitos funcionales de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.1.1.1.1. Aplicación móvil

RU-C-F01			
Título	Pantalla de presentación		
Descripción	Al iniciar la *Aplicación móvil* se mostrará una pantalla de presentación, esta pantalla solo mostrará el logotipo de la aplicación sobre un fondo azul.		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Desechable	**Estabilidad**	No

Tabla 8 - Requisito de capacidad RU-C-F01

RU-C-F02			
Título	Apartados principales		
Descripción	Una vez se deje de mostrar la pantalla de presentación, la *Aplicación móvil* permitirá al usuario seleccionar entre dos apartados principales: • *Control manual* • *Control automático*		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 9 - Requisito de capacidad RU-C-F02

<table>
<tr><th colspan="4">RU-C-F03</th></tr>
<tr><td>Título</td><td colspan="3">Control manual</td></tr>
<tr><td>Descripción</td><td colspan="3">Al seleccionar el usuario el apartado Control manual, la Aplicación móvil permitirá al usuario conectarse y controlar un vehículo de forma remota. Además, también le permitirá observar de forma opcional el entorno que rodea a dicho vehículo.

Para ello, la Aplicación móvil pedirá al usuario la dirección IP del Servidor a conectarse, antes de entrar en este apartado. Una vez indicada, se establecerá conexión con el Servidor a través de dicha IP y permitirá las siguientes acciones al usuario:
• Envío de las siguientes peticiones de movimiento:
○ Girar a la izquierda el vehículo.
○ Girar a la derecha el vehículo.
○ Centrar dirección del vehículo.
○ Hacer avanzar al vehículo.
○ Hacer retroceder al vehículo.
○ Activar cámara del vehículo.
○ Desactivar cámara del vehículo.
○ Girar cámara del vehículo hacia izquierda.
○ Girar cámara del vehículo hacia derecha.
○ Girar cámara del vehículo hacia abajo.
○ Girar cámara del vehículo hacia arriba.
○ Centrar cámara del vehículo.
• Observar el entorno que rodea al vehículo a través de una cámara conectada al vehículo.
Una vez establecida la conexión con el Servidor, la Aplicación móvil será la encargada de realizar de forma automática las siguientes peticiones al Servidor antes de permitir las acciones mencionadas:
• Modo de Control manual.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Alta</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
</table>

Tabla 10 - Requisito de capacidad RU-C-F03

<table>
<tr><th colspan="4">RU-C-F04</th></tr>
<tr><td>Título</td><td colspan="3">Control Automático</td></tr>
<tr><td>Descripción</td><td colspan="3">Al seleccionar el usuario el apartado Control automático, la Aplicación móvil permitirá al usuario la seleccionar entre dos subapartados:
• Nuevo trayecto.
• Ejecutar trayecto.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Alta</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
</table>

Tabla 11 - Requisito de capacidad RU-C-F04

RU-C-F05			
Título	Base de datos		
Descripción	La *Aplicación móvil* utilizará una base de datos alojada en el dispositivo móvil para el almacenamiento de los trayectos.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No

Tabla 12 - Requisito de capacidad RU-C-F05

RU-C-F06			
Título	*Nuevo trayecto*		
Descripción	Al seleccionar el usuario el apartado *Nuevo trayecto,* la *Aplicación móvil* permitirá al usuario la creación y almacenamiento de un nuevo trayecto en la base de datos del dispositivo móvil. Para ello ofrecerá al usuario las siguientes acciones en este apartado: • Especificar los movimientos del trayecto. • Observar un listado de los movimientos de los que está compuesto el trayecto, pudiendo eliminar aquellos que desee. Este listado estará inicialmente vacío. • Especificar las observaciones del trayecto. • Confirmar la creación del trayecto y su posterior almacenamiento en la base de datos.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 13 - Requisito de capacidad RU-C-F06

RU-C-F07			
Título	*Ejecutar trayecto*		
Descripción	Al seleccionar el usuario el apartado *Ejecutar trayecto,* la *Aplicación móvil* mostrará un listado de todos los trayectos almacenados en la base de datos del dispositivo móvil. En dicho listado, el usuario podrá realizar las siguientes acciones en cada uno de los trayectos del listado: • *Realizar trayecto*. • *Editar trayecto*. • *Borrar trayecto*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 14 - Requisito de capacidad RU-C-F07

RU-C-F08			
Título	*Realizar trayecto*		
Descripción	Al seleccionar el usuario la acción *Realizar trayecto* de un trayecto del apartado *Ejecutar trayecto*, la *Aplicación móvil* permitirá al usuario conectarse y enviar a un vehículo el conjunto de movimientos que forman el trayecto seleccionado. Además, también permitirá al usuario observar el entorno del vehículo y el estado de realización de los movimientos, mientras el vehículo los realiza. Para ello, la *Aplicación móvil* pedirá al usuario la dirección IP del *Servidor* a conectarse, antes de realizar la acción *Realizar trayecto*. Una vez indicada, establecerá conexión con el *Servidor* a través de dicha dirección IP y permitirá las siguientes acciones al usuario: • Observar el entorno que rodea al vehículo a través de una cámara conectada al vehículo. • Observar un listado de los movimientos del trayecto a seleccionado, donde se indicará el estado de realización de dichos movimientos. Una vez establecida la conexión con el *Servidor*, la *Aplicación móvil* será la encargada de realizar de forma automática las siguientes peticiones al *Servidor* antes de permitir las acciones mencionadas: • Modo de *Control automático.* • Activar cámara del vehículo. • Envió de trayecto seleccionado.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 15 - Requisito de capacidad RU-C-F08

RU-C-F09			
Título	*Editar trayecto*		
Descripción	Al seleccionar el usuario la acción *Editar trayecto* de un trayecto del apartado *Ejecutar trayecto*, la *Aplicación móvil* permitirá al usuario la modificación en la base de datos del trayecto seleccionado. Para ello la *Aplicación móvil* ofrecerá al usuario las siguientes acciones: • Añadir movimientos al trayecto. • Observar un listado de los movimientos de los que está compuesto el trayecto, pudiendo eliminar aquellos que desee. • Modificar las observaciones del trayecto. • Confirmar la modificación del trayecto y su posterior actualización en la base de datos.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No

Tabla 16 - Requisito de capacidad RU-C-F09

RU-C-F10			
Título	*Eliminar trayecto*		
Descripción	Al seleccionar el usuario la acción *Eliminar trayecto* de un trayecto del apartado *Ejecutar trayecto*, permitirá al usuario eliminar de la base de datos el trayecto seleccionado. Para ello la *Aplicación móvil* ofrecerá al usuario las siguientes acciones: • Confirmación de la eliminación del trayecto y su posterior borrado.		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	No

Tabla 17 - Requisito de capacidad RU-C-F10

RU-C-F11			
Título	Finalización modo de control		
Descripción	Al salir del apartado *Control manual* o al finalizar la acción *Realizar trayecto* en el apartado *Ejecutar trayecto*, la aplicación móvil enviará al *Servidor* las siguientes peticiones de forma automática: ▪ Desactivación de cámara del vehículo si esta fue activada, solo en el caso del apartado *Control manual.* ▪ Petición de finalización de modo de control. ▪ Cierre de conexión con el *Servidor.*		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 18 - Requisito de capacidad RU-C-F11

RU-C-F12			
Título	Espera respuestas peticiones al *Servidor*		
Descripción	Siempre que se realice de forma correcta él envió de una petición al *Servidor*, se esperará de forma indefinida a recibir una respuesta del *Servidor* respecto a la petición, siendo las posibles respuestas: • Respuesta esperada: Indicará que la respuesta se ha realizado de forma correcta. Si se da esta respuesta, se continua de forma normal la ejecución. • Respuesta diferente a la esperada: Toda aquella respuesta que no sea la esperada se considerará como que ha sucedido un error en la realización de la petición. Si se da esta respuesta, se notifica este error al usuario, indicando que petición lo ha causado, y se continua la ejecución.		
Prioridad	Alta	**Verificabilidad**	Baja
Necesidad	Esencial	**Estabilidad**	Si

Tabla 19 - Requisito de capacidad RU-C-F12

3.1.1.1.2. Servidor

RU-C-F13			
Título	Obtención dirección IP del vehículo		
Descripción	Nada más ejecutarse él *Servidor,* este deberá demostrar la dirección IP del dispositivo sobre el que se ejecuta.		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Desechable	**Estabilidad**	Si

Tabla 20 - Requisito de capacidad RU-C-F13

RU-C-F14			
Título	Espera de conexiones de usuarios a través de aplicación móvil		
Descripción	El *Servidor* esperará indefinidamente a que un usuario se conecte a través de la *Aplicación móvil*.		
Prioridad	Alta	**Verificabilidad**	Baja
Necesidad	Esencial	**Estabilidad**	No

Tabla 21 - Requisito de capacidad RU-C-F14

RU-C-F15			
Título	Conexión de usuario desde la aplicación móvil		
Descripción	Una vez conectado a través de la *Aplicación móvil* un usuario al *Servidor*, el usuario deberá de indicar, través de una petición de modo de control, el modo de control escogido para controlar el vehículo. Los modos de control disponibles serán: • *Control manual*. • *Control automático*. El *Servidor* esperará de forma indefinida a esta petición de modo de control.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 22 - Requisito de capacidad RU-C-F15

RU-C-F16			
Título	Selección modo de *Control manual*		
Descripción	Si el usuario conectado envía una petición de control para el modo de *Control manual*, el *Servidor* pasará a modo de *Control manual* e informará del modo de control escogido.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 23 - Requisito de capacidad RU-C-F16

RU-C-F17			
Título	Selección modo de *Control automático*		
Descripción	Si el usuario conectado envía una petición de control para el modo de *Control automático*, el *Servidor* pasará a modo de *Control automático* e informará del modo de control escogido.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 24 - Requisito de capacidad RU-C-F17

RU-C-F18			
Título	Modo de *Control manual*		

Descripción	Cuando el *Servidor* se encuentre en el modo de *Control manual*, deberá poder recibir y realizar las siguientes peticiones realizadas por el usuario:
	• Finalización de modo de control.
	• Activar/Desactivar cámara.
	• Las siguientes peticiones de movimiento:
	  ○ Girar a la izquierda.
	  ○ Girar a la derecha.
	  ○ Centrar dirección del vehículo.
	  ○ Avanzar.
	  ○ Retroceder.
	  ○ Girar cámara izquierda.
	  ○ Girar cámara derecha.
	  ○ Girar cámara abajo.
	  ○ Girar cámara arriba.		

Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 25 - Requisito de capacidad RU-C-F18

RU-C-F19			
Título	Transmisión de imágenes de la cámara		
Descripción	El *Servidor* deberá de ser capaz de transmitir las imágenes capturadas por la cámara cuando esta este activa.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 26 - Requisito de capacidad RU-C-F19

RU-C-F20			
Título	Modo de *Control automático*		
Descripción	En el modo de *Control automático* el *Servidor* deberá de realizar las siguientes acciones: 1. Esperar a la recepción de una petición de activación de cámara por parte del usuario conectado, activando la cámara cuando esta se reciba. 2. Esperar a la recepción del trayecto a ejecutar, almacenándolo cuando este se reciba. 3. Realización de forma autónoma de los movimientos especificados por el trayecto recibido, enviado el estado de realización de cada movimiento según se realice y enviando las peticiones de movimiento que se consideren oportunas al *Controlador*. Si en el punto 1 y 2 se recibe un mensaje que no sea el esperado, se responderá al usuario con un mensaje de error y se seguirá el mismo proceso que si se tratase de una petición de finalización de modo.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 27 - Requisito de capacidad RU-C-F20

RU-C-F21			
Título	Realización de conducción autónoma		
Descripción	La toma de decisiones de la conducción autónoma se deberá de realizar utilizando las imágenes capturadas por una cámara conectada al dispositivo que ejecuta el *Servidor*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 28 - Requisito de capacidad RU-C-F21

RU-C-F22			
Título	Finalización de modo de control		
Descripción	Si el *Servidor* recibe una petición de finalización de modo de control, mientras se encuentra en alguno de los modos de control, este saldrá del modo de control, cerrará la conexión con el usuario y volver a esperar la conexión de un usuario.		
Prioridad	Media	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 29 - Requisito de capacidad RU-C-F22

RU-C-F23			
Título	Respuesta petición de usuario		
Descripción	Aquellas peticiones que estén permitidas en el momento que se reciben en el *Servidor*, se las responderá indicando el existo o el fracaso a la hora de la realización de dichas peticiones. En el caso de que se reciba una petición no conocida o que no está permitida en el momento de su recepción, no se realizará dicha petición y se responderá con un mensaje de error.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 30 - Requisito de capacidad RU-C-F23

RU-C-F24			
Título	Peticiones de movimiento a *Controlador*		
Descripción	El *Servidor* deberá de realizar una petición de movimiento al *Controlador* cuando desee que el vehículo realice alguna de las siguientes acciones: ○ Girar a la izquierda. ○ Girar a la derecha. ○ Centrar dirección del vehículo. ○ Avanzar. ○ Retroceder. ○ Girar cámara izquierda. ○ Girar cámara derecha. ○ Girar cámara abajo. ○ Girar cámara arriba.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No

Tabla 31 - Requisito de capacidad RU-C-F24

3.1.1.1.3. Controlador

RU-C-F25			
Título	Peticiones movimiento *Controlador*		
Descripción	El *Controlador* deberá de reconocer las siguientes peticiones de movimiento: • Girar a la izquierda. • Girar a la derecha. • Centrar dirección del vehículo. • Avanzar. • Retroceder. • Girar cámara izquierda. • Girar cámara derecha. • Girar cámara abajo. • Girar cámara arriba.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 32 - Requisito de capacidad RU-C-F25

RU-C-F26			
Título	Realización y respuesta peticiones de movimiento *Controlador*		
Descripción	Si el *Controlador* recibe una petición de movimiento conocida, llevará acabo el movimiento correspondiente y devolverá una respuesta que indique que se ha recibido la petición de movimiento de forma correcta.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 33 - Requisito de capacidad RU-C-F26

RU-C-F27			
Título	Peticiones no reconocidas		
Descripción	Toda aquella petición que no corresponda con ninguna de las conocidas por el *Controlador*, se responderá con un mensaje de error.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 34 - Requisito de capacidad RU-C-F27

3.1.2. Requisitos de restricción

En este apartado se especificarán los distintos requisitos de usuario de restricción que el sistema debe de cumplir.

3.1.2.1. Requisitos de rendimiento

En este apartado se muestran los requisitos de rendimiento de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.1.2.1.1. Aplicación móvil

RU-R-P01			
Título	Tiempo pantalla de presentación		
Descripción	La pantalla de presentación de la aplicación móvil se mostrará durante 3sg.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Desechable	**Estabilidad**	No

Tabla 35 - Requisito de restricción RU-R-P01

3.1.2.2. Requisitos de seguridad

En este apartado se muestran los requisitos de seguridad de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.1.2.2.1. Aplicación móvil

RU-R-S01			
Título	Error establecimiento de conexión vehículo		
Descripción	Si sucede un error al intentar establecer conexión con el vehículo en el apartado *Control manual* o al realizar la acción de *Realizar trayecto* en el apartado *Ejecutar trayecto*, la *Aplicación móvil* notificará este suceso al usuario y no permitirá la entrada al apartado de *Control manual* o se cancelará la acción de *Realizar trayecto* respectivamente.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 36 - Requisito de restricción RU-R-S01

RU-R-S02			
Título	Error envío petición vehículo		
Descripción	Si sucede un error por perdida de conexión con el *Servidor* al realizarle una petición, la *Aplicación móvil* notificará este error al usuario y se cerrarán todas las conexiones con el *Servidor*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 37 - Requisito de restricción RU-R-S02

3.1.2.2.2. Servidor

RU-R-S03			
Título	Perdida de conexión con *Aplicación móvil*		
Descripción	Si sucede un error por perdida de conexión con el usuario de la *Aplicación móvil*, el *Servidor* realizará las siguientes acciones en el orden que se citan: • Finalizará el modo de control en el cual se encuentre, si se encuentra en alguno. • Cerrará la comunicación con el usuario. • Volverá a la espera de la comunicación con otro usuario.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si

Tabla 38 - Requisito de restricción RU-R-S03

3.1.2.3. Requisitos de implantación

En este apartado se muestran los requisitos de implantación de la aplicación divididos tres apartados distintos, uno por cada componente de la aplicación.

3.1.2.3.1. Aplicación móvil

RU-R-I01			
Título	Versión de Android mínima		
Descripción	La aplicación solo funcionará en dispositivos que dispongan del SO Android con versión 6.0 o superior.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Desechable	**Estabilidad**	No

Tabla 39 - Requisito de restricción RU-R-I01

RU-R-I02			
Título	Conexión misma red que el *Servidor*		
Descripción	El dispositivo móvil desde el que se ejecuta la *Aplicación móvil* deberá estar conectado a la misma red que el *Servidor,* para poder conectarse al vehículo y dar la posibilidad al usuario de poder utilizar todas las funcionalidades de la aplicación.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	No

Tabla 40 - Requisito de restricción RU-R-I02

3.1.2.3.2. Servidor

RU-R-I03			
Título	Red de conexión necesaria		
Descripción	El *Servidor* deberá de estar conectado a una red para que los dispositivos móviles puedan conectarse al vehículo y que el *Servidor* pueda transmitir las imágenes tomadas por la cámara.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No

Tabla 41 - Requisito de restricción RU-R-I03

RU-R-I04			
Título	Disponibilidad de una cámara		
Descripción	El *Servidor* deberá ejecutarse sobre un dispositivo que incorpore o tenga conectada una cámara para un correcto funcionamiento de este componente, así como dar la posibilidad al usuario de poder utilizar todas las funcionalidades de la aplicación.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No

Tabla 42 - Requisito de restricción RU-R-I04

RU-R-I05			
Título	Conexión con *Controlador*		
Descripción	El dispositivo sobre el que se ejecuta el *Servidor* deberá de tener incorporado o conectado él dispositivo sobre el que se ejecuta el *Controlador*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	No

Tabla 43 - Requisito de restricción RU-R-I05

3.1.2.3.3. Controlador

RU-R-I06			
Título	Compilación programa *Controlador*		
Descripción	Se necesitará un Arduino IDE con las librerías del *Shield* Adafruit Motor/Stepper/Servo Shield v1.2 para poder realizar la compilación del programa *Controlador*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si

Tabla 44 - Requisito de restricción RU-R-I06

3.2. Casos de uso

En este apartado se definen los distintos casos de uso de la aplicación. Cada caso de uso describe los pasos que realiza un actor sobre la aplicación para llevar a cabo una funcionalidad de esta. Los casos de uso de la aplicación que se desarrolla en este proyecto son los siguientes:

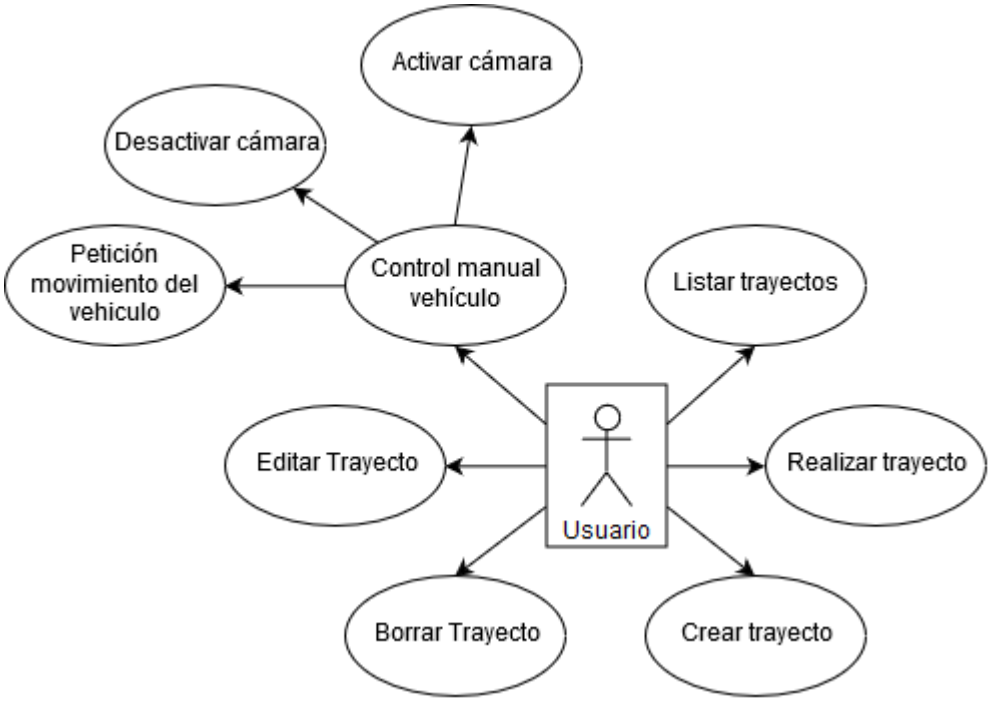

Ilustración 33 - Casos de uso Usuario

Para detallar en profundidad cada caso de uso de la aplicación, se rellanará una tabla con el siguiente formato:

ID	
Actor	
Nombre	
Objetivo	
Pre-condiciones	
Post-condiciones	
Escenario principal	
Escenario alternativo N	

Tabla 45 - Formato tablas casos de uso

- **ID.** Identificador univoco del caso de uso, cada caso de uso tendrá su propio ID que lo identificará con respecto al resto de casos de uso. Este tendrá el siguiente formato:

CU-XX *Estado*

- CU. Indica que se trata de un Caso de Uso.
- XX. Indica el número de caso de uso, este se asignará por orden de creación de los casos de uso.

- **Actor:** Persona, personas o agente externo que interactúa con el sistema.
- **Nombre:** Palabra o conjunto de palabras que indica que indica de forma clara el caso de uso que se está definiendo. Este hará de breve resumen del caso de uso.
- **Objetivo**: Indica que acción pretende conseguir el actor al realizar las acciones especificadas en el caso de uso.
- **Precondiciones**: Condición o condiciones que se deben de cumplir para poder realizar el caso de uso.
- **Postcondiciones**: Aquellas condiciones que una vez realizado el caso de uso se dan en el sistema.
- **Escenario principal**: Serie de pasos que lleva acabo el actor para llevar acabo el caso de uso, pasando de las Precondiciones a las Postcondiciones
- **Escenario alternativo**: Serie de pasos alternativos que puede llevar acabo el actor para realizar el caso de uso. Dependiendo del caso de uso, este campo existirá o no. Si se da más de un caso alternativo, se irán añadiendo campos denominados Escenario alternativo N, donde N es el número de escenario alternativo.

Los casos de uso de la aplicación son los siguientes:

CU-01	
Actor	Usuario de *Aplicación móvil*
Nombre	Listar trayectos
Objetivo	El usuario podrá listar los trayectos almacenados.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta.
Postcondiciones	El usuario observará el listado de todos los trayectos almacenados en la BBDD de la aplicación y dará la posibilidad, en cada elemento del listado, de realizar las acciones *Realizar*, *Editar* y *Borrar*.
Escenario principal	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar Trayecto*. 3. La Aplicación móvil muestra un listado con todos los trayectos almacenados.
Escenario alternativo 1	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar Trayecto*. 3. La Aplicación móvil muestra un mensaje indicando que no hay trayectos almacenados.

Tabla 46 - Caso de uso CU-01

CU-02	
Actor	Usuario de *Aplicación móvil*
Nombre	Borrar trayecto
Objetivo	El usuario podrá eliminar un trayecto almacenado en la BBDD.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Disponer de algún trayecto en la BBDD de la *Aplicación móvil*.
Postcondiciones	Se borrará de la BBDD de la *Aplicación móvil* el trayecto seleccionado por el usuario.
Escenario principal	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Borrar trayecto* del trayecto que desea borrar. 4. El usuario confirma la eliminación del trayecto. 5. Se le informa de la eliminación del trayecto.

Tabla 47 - Caso de uso CU-02

CU-03	
Actor	Usuario de *Aplicación móvil*
Nombre	Editar trayecto
Objetivo	El usuario podrá modificar un trayecto almacenado en la BBDD.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Disponer de algún trayecto en la BBDD de la *Aplicación móvil*.
Postcondiciones	Se actualizará en la BBDD de la *Aplicación móvil* el contenido del trayecto seleccionado por el usuario.
Escenario principal	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Modificar trayecto* del trayecto que desea modificar. 4. Especifica nuevos movimientos para el trayecto si lo considera oportuno, y borrar aquellos movimientos del trayecto que considere oportuno, dejando más de 1 y menos de 1000 movimientos. 5. El usuario pulsa el botón confirmar. 6. El usuario modifica las observaciones si lo considera oportuno. 7. El usuario confirma la modificación del trayecto. 8. Se le informa de la modificación del trayecto.

Escenario alternativo 1	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Modificar trayecto* del trayecto que desea modificar. 4. Especifica nuevos movimientos para el trayecto si lo considera oportuno y borrará aquellos movimientos del trayecto que considere oportuno, dejando más de 999. 5. El usuario pulsa el botón confirmar. 6. La *Aplicación móvil* le informa de que el trayecto puede tener más de 999 movimientos. 7. Especifica los movimientos que desee para dicho trayecto, siendo más de 1 y menos de 1000 movimientos. 8. El usuario pulsa el botón confirmar. 9. El usuario confirma la modificación del trayecto. 10. Se le informa de la modificación del trayecto.
Escenario alternativo 2	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Modificar trayecto* del trayecto que desea modificar. 4. Especifica nuevos movimientos para el trayecto si lo considera oportuno y borrará aquellos movimientos del trayecto que considere oportuno, dejando menos de 1. 5. El usuario pulsa el botón confirmar. 6. La *Aplicación móvil* le informa de que el trayecto no tiene movimientos. 7. Especifica los movimientos que desee para dicho trayecto, siendo más de 1 y menos de 1000 movimientos. 8. El usuario confirma la modificación del trayecto. 9. Se le informa de la modificación del trayecto.

Tabla 48 - Caso de uso CU-03

CU-04	
Actor	Usuario de *Aplicación móvil*
Nombre	Crear trayecto
Objetivo	El usuario podrá crear un nuevo trayecto
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta.
Postcondiciones	Un nuevo trayecto queda almacenado en la BBDD.
Escenario principal	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Nuevo trayecto*. 3. Especifica los movimientos que desee para dicho trayecto, siendo más de 1 y menos de 999 movimientos. 4. El usuario pulsa el botón confirmar. 5. El usuario especifica las observaciones. 6. El usuario confirma la creación del trayecto. 7. Se le informa de la nueva creación del trayecto.
Escenario alternativo 1	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado Nuevo trayecto. 3. No especifica ningún movimiento. 4. El usuario pulsa el botón confirmar. 5. La *Aplicación móvil* le informa de que no ha introducido ningún movimiento. 6. Especifica los movimientos que desee para dicho trayecto, siendo más de 1 y menor que 1000 movimientos. 7. El usuario pulsa el botón confirmar. 8. El usuario especifica las observaciones. 9. El usuario confirma la creación del trayecto. 10. Se le informa de la nueva creación del trayecto.
Escenario alternativo 2	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Nuevo trayecto*. 3. Especifica más de 999 movimientos. 4. El usuario pulsa el botón confirmar. 5. La *Aplicación móvil* le informa de no se pueden superar los 999 movimientos. 6. Elimina los movimientos que desee para dicho trayecto, quedando más de 1 y menos de 1000. 7. El usuario pulsa el botón confirmar. 8. El usuario especifica las observaciones. 9. El usuario confirma la creación del trayecto. 10. Se le informa de la nueva creación del trayecto.

Tabla 49 - Case de uso CU-04

CU-05	
Actor	Usuario de *Aplicación móvil*
Nombre	Realizar trayecto
Objetivo	El usuario podrá enviar un trayecto al *Servidor* y este enviará las peticiones de movimiento necesarias al *Controlador* para que el vehículo lleve a cabo el trayecto de forma autónoma.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Disponer de algún trayecto en la BBDD de la *Aplicación móvil*. • Tener conectado una cámara al dispositivo en el que se está ejecutando el *Servidor*. • Tener el *Servidor* ejecutando sobre un dispositivo conectado a una red. • Tener el dispositivo sobre el que se ejecuta la *Aplicación móvil* conectada a la misma red que el *Servidor*. • Tener el dispositivo sobre el que se ejecuta el *controlador* conectado al dispositivo que ejecuta el *Servidor.*
Postcondiciones	El vehículo habrá realizado el trayecto seleccionado de forma autónoma.
Escenario principal	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Realizar trayecto* del trayecto que desea realizar. 4. El usuario especifica la IP del *Servidor*. 5. La *Aplicación móvil* muestra las imágenes capturadas por la cámara del vehículo y un listado con los movimientos del trayecto y el estado de realización de estos. 6. La *Aplicación móvil* actualiza el estado de los movimientos. 7. La *Aplicación móvil* indica que el trayecto se ha completado.
Escenario alternativo	1. El usuario accede al apartado *Control automático.* 2. El usuario accede al subapartado *Ejecutar trayecto*. 3. El usuario selecciona la acción *Realizar trayecto* del trayecto que desea realizar. 4. El usuario especifica mal la IP del *Servidor*. 5. La Aplicación móvil informa de que no se ha podido establecer la conexión con el *Servidor.* 6. El usuario especifica bien la IP del *Servidor* 7. La *Aplicación móvil* muestra las imágenes capturadas por la cámara del vehículo y un listado con los movimientos del trayecto y el estado de realización de estos. 8. La *Aplicación móvil* actualiza el estado de los movimientos. 9. La *Aplicación móvil* indica que el trayecto se ha completado.

Tabla 50 - Caso de uso CU-05

CU-06	
Actor	Usuario de *Aplicación móvil*
Nombre	Control manual vehículo
Objetivo	El usuario podrá conectarse al *Servidor*, para el posterior envió de peticiones de activación/desactivación de cámara y movimientos del vehículo.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Tener el *Servidor* ejecutando sobre un dispositivo conectado a una red. • Tener el dispositivo sobre el que se ejecuta la *Aplicación móvil* conectada a la misma red que el *Servidor*. • Tener el dispositivo sobre el que se ejecuta el *controlador* conectado al dispositivo que ejecuta el *Servidor.*
Postcondiciones	El usuario podrá enviar peticiones de activación/desactivación de cámara y movimiento del vehículo al *Servidor* conectado.
Escenario principal	1. El usuario accede al apartado *Control manual.* 2. El usuario especifica la IP del *Servidor*. 3. La *Aplicación móvil* muestra la interfaz para el envío de peticiones de activación/desactivación de cámara y movimiento del vehículo.
Escenario alternativo	1. El usuario accede al apartado *Control manual.* 2. El usuario especifica mal la IP del *Servidor*. 3. La Aplicación móvil informa de que no se ha podido establecer la conexión con el *Servidor.* 4. El usuario especifica bien la IP del *Servidor.* 5. La *Aplicación móvil* muestra la interfaz para el envío de peticiones de activación/desactivación de cámara y movimiento del vehículo.

Tabla 51 - Caso de uso CU-06

CU-06	
Actor	Usuario de *Aplicación móvil*
Nombre	Activación cámara
Objetivo	El usuario podrá activar la cámara conectada al *Servidor* y observar las imágenes capturadas por esta a través de la *Aplicación móvil*.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Tener conectado una cámara al dispositivo en el que se está ejecutando el *Servidor*. • Tener el *Servidor* ejecutando sobre un dispositivo conectado a una red. • Tener el dispositivo sobre el que se ejecuta la *Aplicación móvil* conectada a la misma red que el *Servidor*. • Tener el dispositivo sobre el que se ejecuta el *controlador* conectado al dispositivo que ejecuta el *Servidor.* • Haber realizado el caso de uso CU-06. • La cámara conectada al *Servidor* debe estar desactivada.
Postcondiciones	El usuario podrá observar el entorno del vehículo a través de la *Aplicación móvil*.
Escenario principal	1. El usuario pulsa el botón para enviar una petición de activación de la cámara. 2. La *Aplicación móvil* muestra en la interfaz de la aplicación las imágenes capturadas por la cámara conectada al vehículo.

Tabla 52 - Caso de uso CU-07

CU-08	
Actor	Usuario de *Aplicación móvil*
Nombre	Desactivación cámara
Objetivo	El usuario podrá desactivar la cámara conectada al *Servidor* y dejar de observar las imágenes capturadas por esta a través de la *Aplicación móvil*.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Tener conectado una cámara al dispositivo en el que se está ejecutando el *Servidor*. • Tener el *Servidor* ejecutando sobre un dispositivo conectado a una red. • Tener el dispositivo sobre el que se ejecuta la *Aplicación móvil* conectada a la misma red que el *Servidor*. • Tener el dispositivo sobre el que se ejecuta el *controlador* conectado al dispositivo que ejecuta el *Servidor.* • Haber realizado el caso de uso CU-06. • La cámara conectada al *Servidor* debe estar desactivada.
Postcondiciones	El usuario podrá dejar de observar el entorno del vehículo a través de la *Aplicación móvil*.
Escenario principal	1. El usuario pulsa el botón para enviar una petición de desactivación de la cámara y la cámara esta activa. 2. La *Aplicación móvil* deja de mostrar en la interfaz de la aplicación las imágenes capturadas por la cámara conectada al vehículo.

Tabla 53 - Caso de uso CU-08

CU-09	
Actor	Usuario de *Aplicación móvil*
Nombre	Petición movimiento vehículo.
Objetivo	El usuario podrá enviar peticiones de movimiento al *Servidor,* este las enviará directamente al *Controlador* que hará que las lleve a cabo el vehículo.
Precondiciones	• Tener la *Aplicación móvil* instalada. • Tener la *Aplicación móvil* abierta. • Tener conectado una cámara al dispositivo en el que se está ejecutando el *Servidor*. • Tener el *Servidor* ejecutando sobre un dispositivo conectado a una red. • Tener el dispositivo sobre el que se ejecuta la *Aplicación móvil* conectada a la misma red que el *Servidor*. • Tener el dispositivo sobre el que se ejecuta el *controlador* conectado al dispositivo que ejecuta el *Servidor.* • Haber realizado el caso de uso CU-06.
Postcondiciones	El vehículo habrá realizado el movimiento deseado por el usuario.
Escenario principal	1. El usuario pulsa alguno de los botones para enviar una petición de movimiento al vehículo. 2. El usuario deja de pulsar el botón que estaba pulsando.

Tabla 54 - Caso de uso CU-09

3.3. Requisitos de software

En este apartado se definen los requisitos de software, extraídos de los requisitos de usuario, de forma detallada, completa y no ambigua. La clasificación de los requisitos será la misma que se ha utilizado en el apartado dedicado a los Requisitos de usuario.

Los requisitos se recogerán en tablas las cuales tendrán el siguiente formato:

ID			
Título			
Descripción			
Prioridad		**Verificabilidad**	
Necesidad		**Estabilidad**	
RU relacionados			

Tabla 55 - Ejemplo formato Requisito de Software

- **ID:** Identificador univoco del requisito, cada requisito tendrá su propio ID que lo identificará con respecto al resto de requisitos. Este tendrá el siguiente formato:

  RU-X-YZZ

  - *RS:* Indica que es un requisito de software.
  - *X:* Indica de que tipo es el requisito, tomará los siguientes valores:
    - *C:* Si el requisito es de capacidad.
    - *R:* Si el requisito es de restricción.
  - *Y:* Indica en a que subcategoría pertenece el requisito, pudiendo tomar los siguientes valores:
    - *F:* Si el requisito pertenece a la subcategoría de requisitos funcionales.
    - *P:* Si el requisito pertenece a la subcategoría de requisitos de rendimiento.
    - *S:* Si el requisito pertenece a la subcategoría de requisitos de seguridad.
    - *I:* Si el requisito pertenece a la subcategoría de requisito de implantación.
  - *ZZ:* Indica el número de requisito, la asignación de este número dependerá de su orden de creación, categoría y subtipo.
- **Título:** Palabra o conjunto de palabras que resume el contenido del requisito, sin entrar en detalles del contenido del mismo.
- **Descripción:** Explicación detallada de aquello de lo que trata el requisito.
- **Prioridad:** Indica la prioridad que tiene el requisito dentro del proyecto. Los valores que puede tomar este campo son:

    - *Alta***:** Indica el mayor nivel de prioridad, asignándose a aquellos requisitos que se deben de cumplir con mayor celeridad.
    - *Media:* Indica el nivel de prioridad intermedio, se asignará a aquello requisitos que, aun siendo necesarios para el funcionamiento del sistema, no lo son lo suficiente para priorizar su realización.
    - *Baja:* Indica el menor nivel de prioridad, se asignará a aquellos requisitos que puedan ser retrasados.
- **Estabilidad***:* Indica la estabilidad del requisito a través del tiempo, es decir, si el requisito se mantendrá constante durante todo el proyecto o podrá ser modificado. Los valores que puede tomar este campo son:
    - *Si:* Indicará que el requisito es estable, manteniéndose a lo largo de todo el proyecto.
    - *No:* Indicará que el requisito no es estable, pudiéndose dar cambios en el mismo.
- **Necesidad**: Indica como de necesario es un requisito para el proyecto. Los valores que puede tomar este campo son:
    - *Esencial*: Indicará que el requisito es imprescindible para el proyecto, imposibilitando que este pueda ser descartado en un futuro.
    - *Desechable*: Indicará un requisito que no es imprescindible para el proyecto, pudiendo ser descartado en etapas posteriores del desarrollo.
- **Verificabilidad:** Indica el nivel de dificultad que tiene un requisito para ser verificado. Los valores que puede tomar este campo son:
    - *Alta:* Indica el máximo nivel de verificabilidad, indicando que el requisito puede ser verificado de forma sencilla.
    - *Media:* Indica el nivel medio de verificabilidad, indicando que se pueden encontrar algunos problemas a la hora de verificar el requisito.
    - *Baja:* Indica el nivel más bajo de verificabilidad, indicando que verificar el requisito puede ser una tarea complicada de llevar acabo.
- **RU relacionados:** Listado de los requisitos de usuarios relacionados con el requisito que se ha definido.

3.3.1. Requisitos de capacidad

En este aparatado se especificarán los distintos requisitos de software de capacidad que el sistema debe de cumplir.

3.3.1.1. Requisitos funcionales

En este apartado se muestran los requisitos funcionales de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.3.1.1.1. Aplicación móvil

RS-C-F01			
Título	Pantalla de presentación *SplashScreen*		
Descripción	Al iniciar la *Aplicación móvil,* esta mostrará una pantalla de presentación cuyas características serán las siguientes: • La pantalla se denominará *SplashScreen.* • Mostrará el logotipo de la aplicación centrado en mitad de la pantalla, tanto de forma vertical como horizontal. • El *background* será un fondo de color azul. • Se mostrará en orientación vertical. • No mostrará *ActionBar.* • Dara paso de forma automática a la pantalla denominada Pantalla *Principal* sin necesidad de realizar ninguna acción por parte del usuario.		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Desechable	**Estabilidad**	No
RU relacionados	RU-C-F01		

Tabla 56 - Requisito de capacidad RS-C-F01

RS-C-F02			
Título	Pantalla *Principal*		
Descripción	La *Aplicación móvil* dispondrá de una pantalla para seleccionar entre los dos apartados principales de los que dispone la *Aplicación móvil,* las características de la pantalla serán las siguientes: • La pantalla se denominará *Principal.* • Dispondrá de 2 botones, que, al ser pulsados, uno permitirá pasar a la pantalla *ManualControl*, y el otro, a la pantalla *AutomaticControl.* • Se mostrará en orientación vertical. • Se mostrará una *ActionBar* sin *button back* y con el texto “CRV”.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F02		

Tabla 57 - Requisito de capacidad RS-C-F02

RS-C-F03			
Título	Pasar a pantalla *ManualControl* desde pantalla *Principal*		

Descripción	Al pulsar el botón de la pantalla *Principal* que permite pasar a la pantalla *ManualControl*, la *Aplicación móvil* realizará las siguientes acciones, en el orden que se citan, antes de pasar a la pantalla *ManualControl*:
	• Mostrará un cuadro de diálogo de tipo *AlertDialog* con las siguientes características:
	  ○ Title: Selección del servidor.
	  ○ Message: Especifique la dirección IP del servidor en formato a.b.c.d
	  ○ Un campo de texto en el cual el usuario podrá escribir la dirección IP del *Servidor* al que se quiere conectar, sin necesidad de escribirla con el formato especificado en *Message*. En este campo de texto, solo se permitirán los caracteres definidos en el estándar UNICODE, con un máximo de 15 caracteres.
	  ○ Dos botones: Aceptar y Cancelar.
	• Si el usuario pulsa el botón Aceptar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	  ○ Cierre del cuadro de diálogo.
	  ○ Pasará a la pantalla *Control manual* con la dirección IP especificada en el cuadro de diálogo.
	• Si el usuario pulsa el botón Cancelar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	  ○ Cierre del cuadro de dialogo.
	  ○ Se mantendrá en la pantalla *Principal*.		

Prioridad	Media	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F02, RU-C-F03		

Tabla 58 - Requisito de capacidad RS-C-F03

RS-C-F04			
Título	Pasar a pantalla *AutomaticControl* desde pantalla *Principal*		
Descripción	Al pulsar el botón de la pantalla *Principal* que permite pasar a la pantalla *AutomaticControl,* la *Aplicación móvil* pasará a la pantalla *AutomaticControl* sin necesidad de realizar acciones adicionales por parte del usuario.		
Prioridad	Alta	**Verificabilidad**	Si
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F02, RU-C-F04		

Tabla 59 - Requisito de capacidad RS-C-F04

RS-C-F05			
Título	Conexión al *Servidor* en *ManualControl*		

Descripción	Antes de mostrar la interfaz de la pantalla *ManualControl*, se intentará establecer conexión a un *socket* con la dirección IP especificada en la Pantalla *Principal* y el puerto 8002. Dependiendo del resultado de la conexión, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	• Si no se puede establecer la conexión:
	  ○ Mostrará una notificación de tipo *Toast* indicando el motivo del error.
	  ○ Volverá a la *Pantalla principal*.
	• Si se establece la conexión:
	  ○ Enviará una petición de modo de control manual a través de la conexión establecida.
	  ○ Cargará la interfaz de la pantalla *ManualControl.*		

Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F02, RU-C-F03		

Tabla 60 - Requisito de capacidad RS-C-F05

<table>
<tr><th colspan="4">RS-C-F06</th></tr>
<tr><td>Título</td><td colspan="3">Pantalla ManualControl</td></tr>
<tr><td>Descripción</td><td colspan="3">La Aplicación móvil dispondrá de una pantalla para poder llevar acabo el control remoto de un vehículo y dar la posibilidad de observar el entorno en el que se encuentra el vehículo controlado. Las características de esta pantalla serán las siguientes:
La pantalla se denominará ManualControl.
Dispondrá de los siguientes botones para el envío de movimientos al Servidor:
Girar a la izquierda.
Girar a la derecha.
Centrar dirección del vehículo.
Avanzar.
Retroceder.
Mover cámara izquierda.
Mover cámara derecha.
Mover cámara abajo.
Mover cámara arriba.
Dispondrá de un botón de tipo ToggleButton para la activación/desactivación de la cámara.
El background de la pantalla deberá de mostrar las imágenes capturadas por la cámara del vehículo cuando esta haya sido activada mediante la Aplicación móvil, en caso contrario, este será un fondo de color blanco.
Se mostrará en orientación horizontal.
No se mostrará ActionBar.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Alta</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F03</td></tr>
</table>

Tabla 61 - Requisito de capacidad RS-C-F06

RS-C-F07			
Título	Salida de la pantalla *ManualControl*		

Descripción	Al salir de la pantalla de *ManualControl* ya sea por pulsar el *button back* del dispositivo o por cierre de la aplicación, la aplicación deberá de realizar previamente las siguientes acciones en el orden que se citan:
	• Comprobará que la conexión con el *Servidor* sigue activa, dependiendo de esto realizará:
	  ○ Si la conexión sigue activa:
	  ▪ Enviará una petición de desactivación de la cámara si esta había sido activada por medio de la pantalla ManualControl.
	  ▪ Enviará una petición de finalización de modo de control.
	  ▪ Cerrará todas las conexiones con el *Servidor*.
	  ▪ Vuelve a la pantalla *Principal* si se ha pulsado el *Button back* del dispositivo, en caso de haberse cerrado la *Aplicación móvil*, lleva acabo su cierre.
	  ○ Si la conexión no sigue activa:
	  ▪ Cerrará todas las conexiones con el *Servidor*.
	  ▪ Vuelve a la pantalla *Principal* si se ha pulsado el *Button back* del dispositivo, en caso de haberse cerrado la *Aplicación móvil*, se realiza su cierre.		

Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F03, *RU-C-F11*		

Tabla 62 - Requisito de capacidad RS-C-F07

RS-C-F08			
Título	Botones de envío de movimientos *ManualControl*		
Descripción	Los botones para el envío de movimiento al *Servidor* de la pantalla *ManualControl* tendrán el siguiente comportamiento: • Cuando sean pulsados, la *Aplicación móvil* enviará la petición de movimiento correspondiente a dicho botón. • Cuando dejen de ser pulsados, la *Aplicación móvil* enviará la petición de movimiento que finaliza el movimiento que iniciaron al ser pulsados.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F03		

Tabla 63 - Requisito de capacidad RS-C-F08

RS-C-F09			
Título	*ToggleButon* activación/desactivación cámara *ManualControl*		
Descripción	El comportamiento del botón de tipo *Togglebuton* para la activación/desactivación de la cámara de la pantalla *ManualControl* será el siguiente: • Inicialmente se encontrará en estado *unChecked.* • Para cambiar de estado, el usuario deberá de pulsar el botón. • Cuando pase de estado *unChecked* a *isChecked*, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: o Envió de una petición de activación de cámara al *Servidor*. o Se cambiará el *background* de la pantalla *ManualControl* de un fondo blanco a las imágenes capturadas por la cámara conectada al vehículo transmitidas en la dirección web http://(Dirección IP del Servidor):8001. • Cuando pase de estado *isChecked* a *unChecked*, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: o Envió de una petición de desactivación de cámara al *Servidor*. o Se cambiará el *background* de la pantalla *ManualControl* de las imágenes capturadas por la cámara conectada al vehículo a un fondo blanco.		
Prioridad	Media	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F03		

Tabla 64 - Requisito de capacidad RS-C-F09

RS-C-F10			
Título	Pantalla *AutomaticControl*		
Descripción	La *Aplicación móvil* dispondrá de una pantalla para seleccionar entre los dos subapartados de los que dispone el control automático de la *Aplicación móvil,* las características de la pantalla serán las siguientes: • La pantalla se denominará *AutomaticControl.* • Dispondrá de 2 botones, que, al ser pulsados, uno permitirá pasar a la pantalla *NewTray*, y el otro, a la pantalla *ExecuteTray.* • Se mostrará en orientación vertical. • Se mostrará una *ActionBar* con *Button back* y con el texto “Control automático”.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F04*		

Tabla 65 - Requisito de capacidad RS-C-F10

RS-C-F11			
Título	*Button back* del *ActionBar* de pantalla la *AutomaticControl*		
Descripción	El *button back* de la *ActionBar* de la pantalla *AutomaticControl*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* volverá a la pantalla *Principal.*		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Desechable	**Estabilidad**	Si
RU relacionados	*RU-C-F04*		

Tabla 66 - Requisito de capacidad RS-C-F11

RS-C-F12			
Título	Pasar a pantalla *NewTray* desde pantalla *AutomaticControl*		
Descripción	Al pulsar el botón que permite pasar a la pantalla *NewTray* en la pantalla *AutomaticControl,* la *Aplicación móvil* pasará a la pantalla *NewTray* sin necesidad de realizar acciones adicionales por parte del usuario.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F04, RU-C-F06*		

Tabla 67 - Requisito de capacidad RS-C-F12

RS-C-F13			
Título	Pasar a pantalla *ExecuteTray* desde pantalla *AutomaticControl*		
Descripción	Al pulsar el botón que permite pasar a la pantalla *ExecuteTray* en la pantalla *AutomaticControl,* la *Aplicación móvil* pasará a la pantalla *ExecuteTray* sin necesidad de realizar acciones adicionales por parte del usuario.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F04, RU-C-F07*		

Tabla 68 - Requisito de capacidad RS-C-F13

RS-C-F14			
Título	Pantalla *NewTray*		

Descripción	La *Aplicación móvil* dispondrá de una pantalla para la creación de nuevos trayectos y su almacenamiento en la BBDD, las características de la pantalla serán las siguientes:
	• La pantalla se denominará *NewTray.*
	• Dispondrá de los siguientes, botones para la especificación de los movimientos del trayecto:
	  ○ Girar a la izquierda.
	  ○ Girar a la derecha
	  ○ Ir de frente.
	  ○ Parar.
	• Dispondrá de un listado de los movimientos que se han añadido al trayecto. Cada uno de los elementos del listado estará compuesto de:
	  ○ Una imagen representativa del movimiento.
	  ○ Texto que indica que movimiento es.
	  ○ Botón para eliminar el movimiento del listado.
	• El listado de movimientos se encontrará inicialmente vacío.
	• Se mostrará en orientación vertical.
	• Se mostrará una *ActionBar* con *Button back*, con el texto "Nuevo trayecto" y un botón de confirmación de trayecto.		

Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F06*		

Tabla 69 - Requisito de capacidad RS-C-F14

RS-C-F15			
Título	Botones de especificación de movimientos *NewTray*		

Descripción	Los botones para la especificación de movimientos de la pantalla *NewTray*, tendrán el siguiente comportamiento:
	• Cuando sean pulsados, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	  ○ Añadirá el movimiento correspondiente a dicho botón al nuevo trayecto.
	  ○ Añadirá al final del listado de movimientos del nuevo trayecto en la pantalla de *NewTray la* imagen, texto y botón de borrado, del movimiento correspondiente al botón pulsado.
	  ○ Se actualiza el listado de movimientos de *NewTray.*		

Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F06*		

Tabla 70 - Requisito de capacidad RS-C-F15

RS-C-F16			
Título	Botón de borrado de movimiento *NewTray*		
Descripción	El botón de eliminar movimiento de cada uno de los movimientos del listado de movimientos de *NewTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: o Eliminará dicho movimiento del trayecto. o Eliminará la imagen, texto y botón de borrado, de dicho movimiento, del listado de movimientos del nuevo trayecto en la pantalla de *NewTray.* o Se actualiza el listado de movimientos de *NewTray.*		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F06*		

Tabla 71 - Requisito de capacidad RS-C-F16

RS-F-17	
Título	Botón confirmación de trayecto de la pantalla *NewTray*
Descripción	El botón confirmación de trayecto de la *ActionBar* de la pantalla *NewTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: o Comprobará que el nuevo trayecto no está vacío o supera los 999 movimientos: ▪ Si está vacío o supera los 999 movimientos: ❖ No permitirá su almacenamiento. ❖ Notificará al usuario, mediante una notificación de tipo *Toast,* si el trayecto está vacío o supera los 999 movimientos. ❖ Mantendrá al usuario en la pantalla *NewTray*. ▪ Si no está vacío y no se superan los 999 movimientos: ❖ Mostrará un cuadro de dialogo de tipo *AlertDialog* con las siguientes características: ➢ Title: Observaciones. ➢ Message: Indique las observaciones que considere oportunas (Este campo no es obligatorio)

	➢ Un campo de texto en el cual el usuario podrá especificar las observaciones. En este campo de texto se permitirán solo los caracteres en el estándar UNICODE, con un máximo de 1000 caracteres. ➢ Dos botones: Aceptar y Cancelar. ❖ Si el usuario pulsa el botón aceptar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ➢ Cierra el cuadro de diálogo. ➢ Almacena el trayecto en la BBDD. ➢ Notifica al usuario, mediante una notificación de tipo Toast, la correcta creación del nuevo trayecto. ➢ Vuelve a la pantalla *AutomaticControl*. ❖ Si el usuario pulsa el botón cancelar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ➢ Cierra el cuadro de diálogo. ➢ Se mantendrá al usuario en la pantalla *NewTray*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F05, RU-C-F06*		

Tabla 72 - Requisito de capacidad RS-C-F17

RS-C-F18			
Título	*Button back* de *ActionBar* de la pantalla *NewTray*		
Descripción	El *button back* de la *ActionBar* de la pantalla *NewTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ○ Descartará el nuevo trayecto, eliminando los movimientos del trayecto y vaciando el listado de movimientos. ○ Volverá a la pantalla *AutomaticControl.*		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F06*		

Tabla 73 - Requisito de capacidad RS-C-F18

RU-C-F19			
Título	Pantalla *ExecuteTray*		

Descripción	La *Aplicación móvil* dispondrá de una pantalla en la que se listarán todos los trayectos que se encuentre en el dispositivo, pudiendo realizar sobre dichos trayectos las acciones de: Realizar, editar y borrar. Esta pantalla tendrá las siguientes características:
	• La pantalla se denominará *NewTray.*
	• Se mostrará un listado de todos los trayectos almacenados en el dispositivo, este listado tendrá los siguientes campos:
	  ○ ID: Donde se mostrará el ID del trayecto
	  ○ NºMovimientos: Donde se mostrará el número de movimientos del trayecto.
	  ○ Observaciones: Donde se mostrará las observaciones del trayecto.
	  ○ Opciones: Donde se mostrará los siguientes botones:
	  ▪ Botón realizar trayecto.
	  ▪ Botón editar trayecto.
	  ▪ Botón borrar trayecto.
	• Se mostrará en orientación horizontal.
	• Se mostrará una *ActionBar* con *Button back*, con el texto “Ejecutar trayecto”		

Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F07*		

Tabla 74 - Requisito de capacidad RS-C-F19

<table>
<tr><th colspan="4">RS-C-F20</th></tr>
<tr><td>Título</td><td colspan="3">Botón realizar trayecto de la pantalla ExecuteTray</td></tr>
<tr><td>Descripción</td><td colspan="3">El botón realizar trayecto de cada uno de los trayectos de ExecuteTray, tendrá el siguiente comportamiento:
Cuando sea pulsado, la Aplicación móvil realizará las siguientes acciones en el orden que se citan:
Mostrará un cuadro de dialogo de tipo AlertDialog con las siguientes características:
Title: Selección del servidor.
Message: Especifique la dirección IP del servidor en formato a.b.c.d
Un campo de texto en el cual el usuario podrá escribir la dirección IP del Servidor al que se quiere conectar, sin necesidad de escribirla con el formato especificado en Message. En este campo de texto solo se permitirán los caracteres del estándar UNICODE, con un máximo de 15 caracteres.
Dos botones: Aceptar y Cancelar.
Si el usuario pulsa el botón Aceptar, la Aplicación móvil realizará las siguientes acciones en el orden que se citan:
Cierre del cuadro de dialogo.
Pasará a la pantalla ExetcuteTray con la dirección IP especificada por el usuario en el campo de texto del cuadro de dialogo, y el ID del trayecto del listado de trayectos cuyo botón realizar trayecto ha sido pulsado.
Si el usuario pulsa el botón Cancelar, la Aplicación móvil realizará las siguientes acciones en el orden que se citan:
Cierre del cuadro de dialogo.
Se mantendrá en la pantalla ExecuteTray.</td></tr>
<tr><td>Prioridad</td><td>Media</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F07, RU-C-F08</td></tr>
</table>

Tabla 75 - Requisito de capacidad RS-C-F20

RS-C-F21			
Título	Botón editar trayecto de la pantalla *ExecuteTray*		
Descripción	El botón editar trayecto de cada uno de los trayectos de *ExecuteTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* pasará a la pantalla *EditTray* con la información del ID del trayecto del listado de trayectos cuyo botón editar trayecto ha sido pulsado.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F07,* RU-C-F09		

Tabla 76 - Requisito de capacidad RS-C-F21

RS-C-F22			
Título	Botón eliminar trayecto de la pantalla *ExecuteTray*		

Descripción	El botón eliminar trayecto de cada uno de los trayectos de *ExecuteTray*, tendrá el siguiente comportamiento:
	• Cuando sea pulsados, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	  ○ Mostrará un cuadro de dialogo de tipo *AlertDialog* con las siguientes características:
	  ▪ Title: Borrar trayecto.
	  ▪ Message: Esta seguro que quiere borrar el trayecto con ID: X (X es el ID del trayecto del listado de trayectos cuyo botón eliminar trayecto ha sido pulsado).
	  ▪ Dos botones: Aceptar y Cancelar.
	  ○ Si el usuario pulsa el botón aceptar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan:
	  ▪ Se eliminará de la BBDD el trayecto del listado de trayectos cuyo botón eliminar trayecto ha sido pulsado.
	  ▪ Se cierra el cuadro de dialogo.
	  ▪ Se actualiza la tabla de trayectos de *ExecuteTray.*
	  ○ Si el usuario pulsa el botón cancelar, la aplicación móvil cerrará el cuadro de dialogo.		

Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F07,* RU-C-F10		

Tabla 77 - Requisito de capacidad RS-C-F22

RS-C-F23			
Título	*Button back* del *ActionBar* de la pantalla *ExecuteTray*		
Descripción	El *button back* de la *ActionBar* de la pantalla *ExecuteTray* tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* volverá a la pantalla *AutomaticControl.*		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Desechable	**Estabilidad**	No
RU relacionados	*RU-C-F07*		

Tabla 78 - Requisito de capacidad RS-C-F23

RS-C-F24			
Título	Pantalla *EditTray*		

Descripción	La *Aplicación móvil* dispondrá de una pantalla para la modificación de trayectos almacenados en la BBDD y su actualización en dicha BBDD, las características de la pantalla serán las siguientes:
	• La pantalla se denominará *EditTray.*
	• Dispondrá de los siguientes botones para la especificación de nuevos movimientos para el trayecto:
	  ○ Girar a la izquierda.
	  ○ Girar a la derecha
	  ○ Ir de frente.
	  ○ Parar.
	• Dispondrá de un listado de los movimientos de los movimientos de los que se compone el trayecto. Cada uno de los elementos del listado estará compuesto de:
	  ○ Una imagen representativa del movimiento.
	  ○ Texto que indica que movimiento es.
	  ○ Botón para eliminar el movimiento del listado.
	• La obtención de la información del trayecto se realizará a través de una consulta a la BBDD, realizada mediante el ID especificado en la pantalla *ExecuteTray* antes de pasar a esta pantalla.
	• Se mostrará en orientación vertical.
	• Se mostrará una *ActionBar* con *Button back*, con el texto "Editar trayecto" y un botón de confirmación de trayecto.		

Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F09		

Tabla 79 - Requisito de capacidad RS-C-F24

RS-C-F25			
Título	Botones de especificación de nuevos movimientos *EditTray*		
Descripción	Los botones para la especificación de nuevos movimientos de la pantalla *EditTray*, tendrán el siguiente comportamiento: • Cuando sean pulsados, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ○ Añadirá el movimiento correspondiente a dicho botón al nuevo trayecto. ○ Añadirá al final del listado de movimientos del nuevo trayecto en la pantalla de *EditTray la* imagen, texto y botón de borrado, del movimiento correspondiente al botón pulsado. ○ Se actualiza el listado de movimientos de *EditTray.*		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F09		

Tabla 80 - Requisito de capacidad RS-C-F25

RS-C-F26			
Título	Botón de borrado de movimiento *EditTray*		
Descripción	El botón de eliminar movimiento de cada uno de los movimientos del listado de movimientos de *EditTray*, tendrá el siguiente comportamiento: • Cuando sea pulsados, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ○ Eliminará dicho movimiento del trayecto. ○ Eliminará la imagen, texto y botón de borrado, de dicho movimiento, del listado de movimientos del trayecto en la pantalla de *EditTray.* ○ Se actualiza el listado de movimientos de *EditTray.*		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F09		

Tabla 81 - Requisito de capacidad RS-C-F26

RU-C-F27	
Título	Botón confirmación de trayecto de la pantalla *EditTray*
Descripción	El botón confirmación de trayecto de la *ActionBar* de la pantalla *EditTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: o Comprobará que el trayecto no está vacío y no supera los 999 movimientos: ▪ Si está vacío o supera los 999 movimientos: ❖ No permitirá su almacenamiento. ❖ Notificará al usuario, mediante una notificación de tipo *Toast,* si el trayecto está vacío o supera los 999 movimientos. ❖ Mantendrá al usuario en la pantalla *EditTray*. ▪ Si no está vacío y no supera los 999 movimientos: ❖ Mostrará un cuadro de dialogo de tipo *AlertDialog* con las siguientes características: ➢ Title: Observaciones. ➢ Message: Modifique las observaciones actuales como considere oportuno (No es obligatorio). ➢ Un campo de texto en el cual se mostrarán las observaciones actuales y el usuario podrá modificarlas. En este campo de texto se permitirán solo los caracteres en el estándar UNICODE, con un máximo de 1000 caracteres. ➢ Dos botones: Aceptar y Cancelar. ❖ Si el usuario pulsa el botón aceptar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ➢ Cierra el cuadro de diálogo. ➢ Actualiza el trayecto en la BBDD. ➢ Notifica al usuario, mediante una notificación de tipo Toast, la correcta modificación del trayecto.

	➢ Vuelve a la pantalla *ExecuteTray*. ❖ Si el usuario pulsa el botón cancelar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ➢ Cierra el cuadro de diálogo. ➢ Se mantendrá al usuario en la pantalla *EditTray*.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F09		

Tabla 82 - Requisito de capacidad RS-C-F27

RS-C-F28			
Título	*Button back* de *ActionBar* de la pantalla *EditTray*		
Descripción	El *button back* de la *ActionBar* de la pantalla *EditTray*, tendrá el siguiente comportamiento: • Cuando sea pulsado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ○ Descartará los cambios realizados sobre el trayecto que se está modificando. ○ Volverá a la pantalla *ExecuteTray*.		
Prioridad	Baja	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F09		

Tabla 83 - Requisito de capacidad RS-C-F28

RS-C-F29			
Título	Conexión al *Servidor* en *ExecutingTray*		
Descripción	Antes de mostrar la interfaz de la pantalla *ExecutingTray*, se intentará establecer conexión a un *socket* con la dirección IP especificada en la Pantalla *Principal* y el puerto 8002. Dependiendo del resultado de la conexión, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Si no se puede establecer la conexión: ○ Mostrará una notificación de tipo *Toast* indicando el motivo del error. ○ Volverá a la *ExecuteTray*. • Si se establece la conexión: ○ Enviará una petición de modo de control automático a través de la conexión establecida. ○ Enviará una petición de activación de cámara a través de la conexión establecida. ○ Obtendrá la información del trayecto a través de una consulta a la BBDD, realizada mediante el ID especificado en la pantalla *ExecuteTray* antes de intentar pasar a esta pantalla. ○ Enviará una petición de indicación de número de movimientos. ○ Enviará el número de movimientos del trayecto. ○ Enviará los movimientos del trayecto. ○ Cargará la interfaz de la pantalla *ExecutingTray*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F08		

Tabla 84 - Requisito de capacidad RS-C-F29

<table>
<tr><th colspan="4">RS-C-F30</th></tr>
<tr><td>Título</td><td colspan="3">Pantalla ExecutingTray</td></tr>
<tr><td>Descripción</td><td colspan="3">La Aplicación móvil dispondrá de una pantalla para el envío de trayectos almacenados en la BBDD al Servidor para que este los lleve a cabo de forma autónoma, las características de la pantalla serán las siguientes:
La pantalla se denominará ExecutingTray.
En la mitad superior de la pantalla, se mostrarán las imágenes capturadas por la cámara conectada al vehículo transmitidas en la dirección web http://(Dirección IP del Servidor):8001.
En la mitad inferior de la pantalla, se mostrará un listado con los movimientos que componen al trayecto que se ha enviado al Servidor. Cada uno de los movimientos del listado tendrá los siguientes elementos:
Una imagen representativa del movimiento.
Texto que indica que movimiento es.
Imagen que indica el estado de realización del movimiento. Los estados posibles son:
En realización.
No realizado.
Realizado de forma correcta.
No se ha podido llevar acabo su realización.
No reconocido.
Se mostrará en orientación vertical.
Se mostrará una ActionBar con Button back, con el texto “Ejecutando trayecto…”.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>No</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F08</td></tr>
</table>

Tabla 85 - Requisito de capacidad RS-C-F30

<table>
<tr><th colspan="4">RS-C-F31</th></tr>
<tr><td>Título</td><td colspan="3">Cambio estado de movimientos en pantalla ExecutingTray</td></tr>
<tr><td>Descripción</td><td colspan="3">En la pantalla ExecutingTray, para cada uno de los movimientos del listado de movimientos del trayecto, comenzando por él primero, la Aplicación móvil realizará las siguientes acciones en el orden que se citan:
Modifica la imagen que indica el estado del movimiento a “En realización”.
Esperará de forma indefinida a la respuesta del Servidor sobre el estado del movimiento. Una vez recibida la respuesta, realizará las siguientes acciones en el orden que se citan:
Comprobará de que tipo es la respuesta:
Si la respuesta es que el movimiento se ha realizado de forma correcta:
Modifica la imagen que indica el estado de movimiento a “Realizado de forma correcta”.
Si la respuesta es que el movimiento no se ha podido llevar acabo:
Modifica la imagen que indica el estado de movimiento a “No se ha podido llevar acabo su realización”.
Si la respuesta es un mensaje de error:
Mediante una notificación de tipo Toast indica el suceso.
Lleva acabo las mismas acciones que si se pulsara el button back del dispositivo.
Si la respuesta no se reconoce:
Modifica la imagen que indica el estado de movimiento a “No reconocido”.
Se pasa al siguiente movimiento del listado.</td></tr>
<tr><td>Prioridad</td><td>Media</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>No</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F08</td></tr>
</table>

Tabla 86 - Requisito de capacidad RS-C-F31

RS-C-F32			
Título	Finalización de trayecto en pantalla *ExecutingTray*		
Descripción	En la pantalla *ExecutingTray, c*uando el *Servidor* informe del estado de realización del último movimiento, independientemente de cuál sea dicho estado, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Mostrará un cuadro de dialogo de tipo *AlertDialog* con las siguientes características: ○ Title: Trayecto finalizado. ○ Message: El trayecto ha sido finalizado, observe el listado movimientos para comprobar que movimiento se han realizado de forma correcta y cuáles no. ○ Un botón: Aceptar • Si el usuario pulsa el botón aceptar, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: ○ Cierra el cuadro de diálogo.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F08		

Tabla 87 - Requisito de capacidad RS-C-F32

<table>
<tr><th colspan="4">RS-C-F33</th></tr>
<tr><td>Título</td><td colspan="3">Salida de pantalla ExecutingTray</td></tr>
<tr><td>Descripción</td><td colspan="3">Al salir de la pantalla de ExecutingTray ya sea por pulsar el button back, del dispositivo o del ActionBar de esta pantalla, o por cierre de la aplicación, la aplicación deberá de realizar previamente las siguientes acciones en el orden que se citan:
Comprobará que la conexión con el Servidor sigue activa, dependiendo de esto realizará:
Si la conexión sigue activa:
Enviará una petición de finalización de modo de control.
Cerrará todas las conexiones con el Servidor.
Vuelve a la pantalla ExecuteTray si se ha pulsado el Button back del dispositivo o del ActionBar de esta pantalla. En caso de haberse cerrado la Aplicación móvil, lleva acabo su cierre.
Si la conexión no sigue activa:
Cerrará todas las conexiones con el Servidor.
Vuelve a la pantalla ExecuteTray si se ha pulsado el Button back del dispositivo o del ActionBar de esta pantalla. En caso de haberse cerrado la Aplicación móvil, se lleva a cabo su cierre.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>No</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F08, RU-C-F11, RU-C-F12</td></tr>
</table>

Tabla 88 - Requisito de capacidad RS-C-F33

<table>
<tr><th colspan="4">RS-C-F34</th></tr>
<tr><td>Título</td><td colspan="3">Respuestas recibidas en ExecutingTray desde Servidor</td></tr>
<tr><td>Descripción</td><td colspan="3">Las respuestas reconocidas por la Aplicación móvil en la pantalla ExecutingTray, respecto al estado de los movimientos enviados desde el Servidor, siguen el siguiente formato:
<table>
<tr><th>Estado de movimiento</th><th>Respuesta</th></tr>
<tr><td>Realizado de forma correcta</td><td>MOV: OK\n</td></tr>
<tr><td>El movimiento no se ha podido realizar</td><td>MOV: NO\n</td></tr>
<tr><td>Mensaje de error</td><td>MSG: ERR\n</td></tr>
<tr><td>Respuesta no reconocida</td><td>Cualquier otro mensaje</td></tr>
</table>
Tabla 89 - Formato respuestas estado de movimientos</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>No</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F08</td></tr>
</table>

Tabla 90 - Requisito de capacidad RS-C-F34

RS-C-F35			
Título	*Button back* del dispositivo en pantallas con *ActionBar*		
Descripción	Si se pulsa el *button back* del dispositivo en una pantalla que disponga de *ActionBar* con *button back,* el *button back* del dispositivo tendrá el mismo comportamiento que el *button back* de la *ActionBar*.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Desechable	**Estabilidad**	no
RU relacionados	RU-C-F02, RU-C-F04, RU-C-F06, RU-C-F07, RU-C-F08		

Tabla 91 - Requisito de capacidad RS-C-F35

<table>
<tr><th colspan="4">RS-C-F36</th></tr>
<tr><td>Título</td><td colspan="3">Espera respuestas peticiones al Servidor</td></tr>
<tr><td>Descripción</td><td colspan="3">Siempre que se realice de forma correcta él envió de una petición al Servidor, se esperará de forma indefinida a recibir una respuesta del Servidor respecto a la petición, siendo las posibles respuestas:
Respuesta esperada: Indicará que la respuesta se ha realizado de forma correcta. Si se da esta respuesta, se continua de forma normal la ejecución.
Respuesta diferente a la esperada: Toda aquella respuesta que no sea la esperada se considerará como que ha sucedido un error en la realización de la petición. Si se da esta respuesta, se notifica este error al usuario utilizando una notificación de tipo Toast, indicando que petición lo ha causado, y se continua la ejecución.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Baja</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F12</td></tr>
</table>

Tabla 92 - Requisito de capacidad RS-C-F36

<table>
<tr><th colspan="4">RS-C-F37</th></tr>
<tr><td>Título</td><td colspan="3">Formato de peticiones y respuestas esperadas al Servidor</td></tr>
<tr><td>Descripción</td><td colspan="3">Las peticiones enviadas por la Aplicación móvil al Servidor y sus correspondientes respuestas esperadas serán de 9 bytes y seguirán el siguiente formato:
<table>
<tr><th>Modo de control</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Manual</td><td>MOD: MAN\n</td><td>MOD: OK\n</td></tr>
<tr><td>Automático</td><td>MOD: AUT\n</td><td>MOD: OK\n</td></tr>
<tr><td>Finalización</td><td>MOD: OFF\n</td><td>MOD: OK\n</td></tr>
</table>
Tabla 93 - Formato petición y respuesta esperada selección modo de control
<table>
<tr><th>Cámara</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Activar</td><td>CAM: ON\n</td><td>CAM: OK\n</td></tr>
<tr><td>Desactivar</td><td>CAM: OFF\n</td><td>CAM: OK\n</td></tr>
<tr><td>Girar izquierda</td><td>HOR: DER\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar derecha</td><td>HOR: IZQ\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar izquierda</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar derecha</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar arriba</td><td>VER: ARB\n</td><td>VER: OK\n</td></tr>
<tr><td>Girar abajo</td><td>VER: ABJ\n</td><td>VER: OK\n</td></tr>
<tr><td>Parar girar arriba</td><td>VER: STP\n</td><td>VER: OK\n</td></tr>
</table></td></tr>
</table>

Parar girar abajo	VER: STP\n	VER: OK\n
Centrar	CAM: CEN\n	CAM: OK\n

Tabla 94 - Formato petición y respuesta peticiones cámara

Vehículo	Petición	Respuesta
Avanzar	MTR: AVZ\n	MTR: OK\n
Retroceder	MTR: RTD\n	MTR: OK\n
Parar	MTR: STP\n	MTR: OK\n
Girar izquierda	DIR: IZQ\n	DIR: OK\n
Girar derecha	DIR: DER\n	DIR: OK\n
Parar girar derecha	DIR: STP\n	DIR: OK\n
Parar girar izquierda	DIR: STP\n	DIR: OK\n
Centrar dirección	DIR: CEN\n	DIR: OK\n

Tabla 95 - Formato petición y respuesta esperada movimientos vehículo

Trayecto	Petición	Respuesta
Indicación envió de Número de movimientos	MOV: NUM\n	MOV: OK\n
Número de movimientos Menor que 10	00X (X es el número de movimientos)	MOV: OK\n
Número de movimientos entre 10 y 99	0XX (X es el número de movimientos)	MOV: OK\n
Número de movimientos entre 100 y 999	XXX (X es el número de movimientos)	MOV: OK\n
Movimiento, girar hacia la izquierda	MOV: IZQ\n	MOV: OK\n
Movimiento, girar haciala derecha	MOV: DER\n	MOV: OK\n
Movimiento, continuar de frente	MOV: FTR\n	MOV: OK\n
Movimiento, parar	MOV: STP\n	MOV: OK\n

Tabla 96 - Formato petición y respuesta esperada especificación trayectos

Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F03, RU-C-F08, RU-C-F12		

Tabla 97 - Requisito de capacidad RS-C-F36

RS-C-F38			
Título	La *Aplicación móvil* dispondrá de un BBDD		
Descripción	La *Aplicación móvil* dispondrá de una base de datos denominada *Trayectos* alojada en el dispositivo móvil sobre el que se ejecuta.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F05		

Tabla 98 - Requisito de capacidad RS-C-F38

RS-C-F39			
Título	Tablas de la BBDD *Trayectos* de la Aplicación móvil		
Descripción	La base de datos *Trayectos* de la *Aplicación móvil* dispondrá de las siguientes tablas: • *trayectos*		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F05		

Tabla 99 - Requisito de capacidad RS-C-F39

RS-C-F40			
Título	Tabla *trayectos* de la BBDD *Trayectos* de la Aplicación móvil		
Descripción	La tabla *trayectos* de la BBDD *Trayectos* de la *Aplicación móvil*, se utilizará para el almacenamiento de trayectos para su posterior ejecución. Los campos de esta tabla son: • ID: Campo de tipo entero, cuyo valor asignará de forma automática la BBDD a cada una de las tuplas que se inserten en la tabla *trayectos*. Este campo será la clave primaria de la tabla de *trayectos.* • Num_Movimientos: Campo de tipo entero que no podrá tomar valores superiores a 999 ni el valor NULL. • Movimientos: Cadena de caracteres *ASCII,* que no podrá tomar el valor NULL. • Observaciones: Cadena de caracteres UNICODE con una longitud máxima de 1000.		
Prioridad	Alta	**Verificabilidad**	Baja
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F05		

Tabla 100 - Requisito de capacidad RS-C-F40

3.3.1.1.2. Servidor

RS-C-F41			
Título	Obtención e impresión de dirección IP del *Servidor*		
Descripción	Nada más ejecutarse el *Servidor*, este obtendrá la IP del dispositivo sobre el que se está ejecutando y la imprimirá por pantalla.		
Prioridad	Baja	**Verificabilidad**	Baja
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F13		

Tabla 101 - Requisito de capacidad RS-C-F41

RS-C-F42			
Título	Conexiones usuarios *Aplicación móvil* al *Servidor*		
Descripción	El Servidor dispondrá de un *socket* para que los usuarios de la *Aplicación móvil* puedan conectarse a él. Este socket tendrá las siguientes características: • Dirección IP: La dirección IP obtenida al comenzar a ejecutarse el *Servidor*. • Puerto: 8002.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	RU-C-F14		

Tabla 102 - Requisito de capacidad RS-C-F42

RS-C-F43			
Título	Espera de conexiones de usuarios de la *Aplicación móvil*		
Descripción	El *Servidor* esperará de forma indefinida a que un usuario de la *Aplicación móvil* se conecte a través del *socket* establecido para ello.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F14		

Tabla 103 - Requisito de capacidad RS-C-F43

RS-C-F44			
Título	Conexión con usuario de la *Aplicación móvil*		
Descripción	Si un usuario de la *Aplicación móvil* se conecta al del *socket Servidor* establecido para ello, el *Servidor* establecerá una conexión con dicho usuario y dejará de esperar nuevas conexiones en el *socket*.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	RU-C-F14		

Tabla 104 - Requisito de capacidad RS-C-F44

RS-C-F45			
Título	Primera petición al *Servidor*		
Descripción	Nada más establecerse la conexión con el usuario de la *Aplicación móvil*, el *Servidor* esperará de forma indefinida a que este le envié una petición de 9 bytes. Una vez recibida la petición el Servidor realizará las siguientes acciones en el orden que se citan: • Comprobará que la petición recibida es una petición de modo de control: ○ Si la petición recibida es de modo de control manual, "MOD: MAN\n": ▪ Envía una la respuesta "MOD: OK\n". ▪ Ejecuta la función que realiza el *Control manual*. ○ Si la petición recibida es de modo de control manual, es "MOD: AUT\n": ▪ Envía una la respuesta "MOD: OK\n". ▪ Ejecuta la función que realiza el *Control automático*. ○ Si no es una petición de modo de control: ▪ Independientemente de lo que contenga la petición, la ignorará y volverá a esperar una petición de 9 bytes.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F15, RU-C-F16, RU-C-F17, RU-C-F23*		

Tabla 105 - Requisito de capacidad RS-C-F45

RS-C-F46			
Título	Ejecución función *Control manual*		
Descripción	Cuando el *Servidor* se encuentre ejecutando la función de *Control manual*, realizará las siguientes acciones en el orden que se citan: • Imprimirá por pantalla que se ha escogido el modo de *Control manual*. • Esperará de forma indefinida peticiones de 9 bytes desde la conexión establecida con el usuario de la *Aplicación móvil*. Cuando reciba una petición realizará las siguientes acciones en el orden que se citan: ○ Comprobará el contenido de la petición. ○ Imprimirá por pantalla el contenido de la petición. ○ Procesará dicha petición, realizando las acciones que se consideren oportunas. ○ Volverá a esperar de forma indefinida peticiones de 9 bytes.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F16, RU-C-F18*		

Tabla 106 - Requisito de capacidad RS-C-F46

RS-C-F47			
Título	Procesamiento petición activación de cámara en *Control manual*		
Descripción	Si durante la ejecución de la función *Control manual*, la petición recibida por el *Servidor* es de activación de cámara, "CAM: ON\n", el *Servidor* realizará las siguientes acciones en el orden que se citan, para su procesamiento: • Si la cámara no estaba activa: ○ Lanzará un proceso que se encargará de: ▪ Capturar imágenes a través de la cámara conectada al vehículo con tamaño 640x360 ▪ Transmisión de las imágenes a través de la dirección http://(Dirección IP obtenida al ejecutar el servidor):8001. ○ Envía la respuesta "CAM: OK\n" si no sucede ningún error al lanzar el proceso. ○ Envía la respuesta "MSG: ERR\n" si sucede algún error al lanzar el proceso. • Si la cámara ya estaba activa: ○ Se ignora la petición.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	*RU-C-F18, RU-C-F19, RU-C-F23*		

Tabla 107 - Requisito de capacidad RS-C-F47

RS-C-F48			
Título	Procesamiento petición desactivación de cámara en *Control manual*		

Descripción	Si durante la ejecución de la función *Control manual*, la petición recibida por el *Servidor* es de desactivación de cámara, "CAM: OFF\n", el *Servidor* realizará las siguientes acciones en el orden que se citan, para su procesamiento:
	• Si la cámara estaba activa:
	  ○ Finaliza el proceso que se encargaba de la captura de las imágenes a través de la cámara conectada al vehículo y su posterior transmisión.
	  ○ Envía la respuesta "CAM: OK\n" si no sucede ningún error al finalizar el proceso.
	  ○ Envía la respuesta "MSG: ERR\n" si sucede algún error al lanzar finalizar el proceso.
	• Si la cámara ya estaba activa:
	  ○ Se ignora la petición.		

Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	No
RU relacionados	*RU-C-F18, RU-C-F19, RU-C-F23*		

Tabla 108 - Requisito de capacidad RS-C-F48

RS-C-F49			
Título	Procesamiento petición finalización de modo de control en *Control manual*		
Descripción	Si durante la ejecución de la función *Control manual*, la petición recibida por el *Servidor* es de finalización de modo de control, "MOD: OFF\n", el *Servidor* se realizará las siguientes acciones en el orden que se citan, para su procesamiento: • Envía la respuesta "MOD: OK\n". • Finaliza la función *Control manual*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F18, RU-C-F22, RU-C-F23*		

Tabla 109 - Requisito de capacidad RS-C-F49

RS-C-F50			
Título	Procesamiento del resto de peticiones en *Control manual*		
Descripción	Si durante la ejecución de la función *Control manual*, el *Servidor* recibe una petición que no sea: • Activación de cámara, “CAM: ON\n”. • Desactivación de cámara, “CAM: OFF\n”. • Finalización de modo de control, “MOD: OFF\n”. El *Servidor* realizará las siguientes acciones en el orden que se citan: • Enviará directamente al *Controlador* la petición recibida sin procesar. • Enviará a través de la conexión establecida con el usuario de la *Aplicación móvil,* la respuesta recibida por el *Controlador* al realizar la petición.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F18, RU-C-F23*		

Tabla 110 - Requisito de capacidad RS-C-F51

RS-C-F51			
Título	Finalización función *Control manual* o *Control automático*		
Descripción	Al finalizarse la función que ejecuta el *Control manual* o la que ejecuta el *Control automático*, el *Servidor* cerrará la conexión con el usuario de la *Aplicación móvil* con el que mantenía una conexión y vuelve a esperar la conexión de un usuario de la *Aplicación móvil*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F18, RU-C-F20, RU-C-F22*		

Tabla 111 - Requisito de capacidad RS-C-F51

RS-C-F52	
Título	Ejecución función *Control automático*
Descripción	Cuando el *Servidor* se encuentre ejecutando la función de *Control automático*, realizará las siguientes acciones en el orden que se citan: • Imprimirá por pantalla que se ha escogido el modo de *Control automático*. • Esperará de forma indefinida una petición de activación de cámara, “CAM: ON\n”. Una vez recibida, realizará las siguientes acciones en el orden que se citan: ○ Lanzará un proceso que se encargará de:

<table>
<tr><td></td><td colspan="3">

Captura de imágenes a través de la cámara conectada al vehículo con tamaño 320x240.
Almacenamiento de las imágenes capturadas.
Transmisión de las imágenes a través de la dirección http://(Dirección IP obtenida al ejecutar el servidor):8001.

Envía la respuesta “CAM: OK\n” si no sucede ningún error al lanzar el proceso.
Envía la respuesta “MSG: ERR\n” si sucede algún error al lanzar el proceso.

Esperará de forma indefinida una petición de indicación de número de movimientos de un trayecto, “MOV: NUM\n”. Una vez recibida, realizará las siguientes acciones:

Envía la respuesta “MOV: OK\n”.

Espera de forma indefinida una petición de número de movimientos de un trayecto de 3 bytes. Una vez recibida, realizará las siguientes acciones en el orden que se citan:

Almacena el número de movimientos del trayecto.
Envía la respuesta “MOV: OK\n”.

Espera de forma indefinida por tantas peticiones de especificación de movimiento de un trayecto como movimientos tenga el trayecto. Por cada petición recibida realizará las siguientes acciones en el orden que se citan:

Comprobará que es una petición de especificación de movimiento, “MOV: IZQ\n”, “MOV: DER\n”, “MOV: STP\n” o “MOV: FTR\n”:

Si es una petición de especificación de movimiento del trayecto:

Almacenará el movimiento.
Enviará la respuesta “MOV: OK\n”

Si no es una petición de especificación de especificación de movimiento:

Enviará la respuesta “MOV: ERR\n”
Finalizará la función Control automático.

Realiza los movimientos del trayecto de forma autónoma (Se especifica en un requisito aparte).

</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Alta</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F16, RU-C-F19, RU-C-F20, RU-C-F23</td></tr>
</table>

Tabla 112 - Requisito de capacidad RS-C-F52

<table>
<tr><th colspan="4">RS-C-F53</th></tr>
<tr><td>Título</td><td colspan="3">Conducción autónoma en Control automático</td></tr>
<tr><td>Descripción</td><td colspan="3">Para la conducción autónoma del vehículo en la ejecución de la función de Control automático, el Servidor realizará las siguientes acciones en el orden que se citan:
Toma de la última imagen capturada y almacenada por el proceso encargado de la toma de imágenes mediante el uso de la cámara del vehículo. El archivo de la imagen deberá de: Tener el tamaño 320x240, haber sido tomada hace menos de un segundo y no haber sido procesada anteriormente.
Clasifica dicha imagen en una de las siguientes categorías:
ATRAS
DELANTE
DERECHA
DERECHA-90
DERECHA-DELANTE
IZQUIERDA
IZQUIERDA-90
IZQUIERDA-DELANTE
IZQUIERDA-DERECHA
IZQUIERDA-DERECHA-90
IZQUIERDA-DERECHA-DELANTE
Dependiendo de la categoría de la imagen y el movimiento actual que se quiera realizar, se realizan las peticiones necesarias al Controlador para llevar acabo el movimiento y se enviará el estado del movimiento actual a la Aplicación móvil. Los estados enviados serán:
El movimiento se ha realizado de forma correcta: “MOV: OK\n”
El movimiento no se ha podido realizar: “MOV: NO\n”
Mensaje error: “MSG: ERR\n”
Se pasa al siguiente movimiento del trayecto y se repite el proceso. Si el trayecto no dispone de más movimientos, el Servidor da por finalizada la conducción autónoma y espera de forma indefinida a una petición de finalización de modo de control.</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Baja</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F20, RU-C-F21</td></tr>
</table>

Tabla 113 - Requisito de capacidad RS-C-F53

RS-C-F54			
Título	Procesamiento petición finalización de modo de control en *Control automático*		
Descripción	Si durante la ejecución de la función *Control automático*, la petición recibida por el *Servidor* es de finalización de modo de control, "MOD: OFF\n", el *Servidor* se realizará las siguientes acciones en el orden que se citan, para su procesamiento: • Se envía la respuesta "MOD: OK\n". • Se finaliza la conducción autónoma. • Se finaliza el proceso que se encargaba de la captura de las imágenes a través de la cámara conectada al vehículo y su posterior transmisión. • Se finaliza la función *Control automático*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F20, RU-C-F22, RU-C-F23*		

Tabla 114 - Requisito de capacidad RS-C-F54

RS-C-F55			
Título	Procesamiento del resto de peticiones en conducción autónoma en *Control automático*		
Descripción	Si durante la realización de la conducción autónoma de la función *Control automático*, el *Servidor* recibe una petición que no sea: • Finalización de modo de control, "MOD: OFF\n". El *Servidor* realizará las siguientes acciones en el orden que se citan: • Enviará la respuesta "MSG: ERR\n" • Continua la conducción autónoma.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F20, RU-C-F23*		

Tabla 115 - Requisito de capacidad RS-C-F55

<table>
<tr><th colspan="4">RS-C-F56</th></tr>
<tr><td>Título</td><td colspan="3">Peticiones y respuestas en por Servidor al Controlador</td></tr>
<tr><td>Descripción</td><td colspan="3">El formato de las peticiones realizadas por el Servidor, para que el vehículo realice algún movimiento, al Controlador y las respuestas esperadas a dichas peticiones será el siguiente:
<table>
<tr><th>Cámara</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Girar izquierda</td><td>HOR: DER\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar derecha</td><td>HOR: IZQ\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar izquierda</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar derecha</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar arriba</td><td>VER: ARB\n</td><td>VER: OK\n</td></tr>
<tr><td>Girar abajo</td><td>VER: ABJ\n</td><td>VER: OK\n</td></tr>
<tr><td>Parar girar arriba</td><td>VER: STP\n</td><td>VER: OK\n</td></tr>
<tr><td>Parar girar abajo</td><td>VER: STP\n</td><td>VER: OK\n</td></tr>
<tr><td>Centrar</td><td>CAM: CEN\n</td><td>CAM: OK\n</td></tr>
</table>
Tabla 116 - Formato petición y respuesta peticiones cámara Servidor
<table>
<tr><th>Vehículo</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Avanzar</td><td>MTR: AVZ\n</td><td>MTR: OK\n</td></tr>
<tr><td>Retroceder</td><td>MTR: RTD\n</td><td>MTR: OK\n</td></tr>
<tr><td>Parar</td><td>MTR: STP\n</td><td>MTR: OK\n</td></tr>
<tr><td>Girar izquierda</td><td>DIR: IZQ\n</td><td>DIR: OK\n</td></tr>
<tr><td>Girar derecha</td><td>DIR: DER\n</td><td>DIR: OK\n</td></tr>
<tr><td>Parar girar derecha</td><td>DIR: STP\n</td><td>DIR: OK\n</td></tr>
<tr><td>Parar girar izquierda</td><td>DIR: STP\n</td><td>DIR: OK\n</td></tr>
<tr><td>Centrar dirección</td><td>DIR: CEN\n</td><td>DIR: OK\n</td></tr>
</table>
Tabla 117 - Formato petición y respuesta esperada movimientos Servidor</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F24</td></tr>
</table>

Tabla 118 - Requisito de capacidad RS-C-F56

3.3.1.1.3. Controlador

<table>
<tr><th colspan="4">RS-C-F57</th></tr>
<tr><td>Título</td><td colspan="3">Peticiones y respuestas Controlador</td></tr>
<tr><td>Descripción</td><td colspan="3">El formato de las peticiones reconocidas por el Controlador y las respuestas realizadas a dichas peticiones será el siguiente:
<table>
<tr><th>Cámara</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Girar izquierda</td><td>HOR: DER\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar derecha</td><td>HOR: IZQ\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar izquierda</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Parar girar derecha</td><td>HOR: STP\n</td><td>HOR: OK\n</td></tr>
<tr><td>Girar arriba</td><td>VER: ARB\n</td><td>VER: OK\n</td></tr>
<tr><td>Girar abajo</td><td>VER: ABJ\n</td><td>VER: OK\n</td></tr>
<tr><td>Parar girar arriba</td><td>VER: STP\n</td><td>VER: OK\n</td></tr>
<tr><td>Parar girar abajo</td><td>VER: STP\n</td><td>VER: OK\n</td></tr>
<tr><td>Centrar</td><td>CAM: CEN\n</td><td>CAM: OK\n</td></tr>
</table>
Tabla 119 - Formato petición y respuesta peticiones cámara Controlador
<table>
<tr><th>Vehículo</th><th>Petición</th><th>Respuesta</th></tr>
<tr><td>Avanzar</td><td>MTR: AVZ\n</td><td>MTR: OK\n</td></tr>
<tr><td>Retroceder</td><td>MTR: RTD\n</td><td>MTR: OK\n</td></tr>
<tr><td>Parar</td><td>MTR: STP\n</td><td>MTR: OK\n</td></tr>
<tr><td>Girar izquierda</td><td>DIR: IZQ\n</td><td>DIR: OK\n</td></tr>
<tr><td>Girar derecha</td><td>DIR: DER\n</td><td>DIR: OK\n</td></tr>
<tr><td>Parar girar derecha</td><td>DIR: STP\n</td><td>DIR: OK\n</td></tr>
<tr><td>Parar girar izquierda</td><td>DIR: STP\n</td><td>DIR: OK\n</td></tr>
<tr><td>Centrar dirección</td><td>DIR: CEN\n</td><td>DIR: OK\n</td></tr>
</table>
Tabla 120 - Formato petición y respuesta esperada vehículo Controlador</td></tr>
<tr><td>Prioridad</td><td>Alta</td><td>Verificabilidad</td><td>Media</td></tr>
<tr><td>Necesidad</td><td>Esencial</td><td>Estabilidad</td><td>Si</td></tr>
<tr><td>RU relacionados</td><td colspan="3">RU-C-F25, RU-C-F26</td></tr>
</table>

Tabla 121 - Requisito de capacidad RS-C-F5

RS-C-58			
Título	Respuesta peticiones de movimiento no reconocidas *Controlador*		
Descripción	Si el *Controlador* recibe una petición de movimiento no reconocida, enviará la respuesta “MSG: ERR\n”.		
Prioridad	Media	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F27*		

Tabla 122 - Requisito de capacidad RS-C-F58

RS-C-F59			
Título	Canal envió peticiones y respuestas *Controlador*		
Descripción	El envío de peticiones al *Controlador* y las correspondientes respuestas que este realice, se harán a través del puerto serial del dispositivo sobre el que se ejecute el *Controlador*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F25, RU-C-F26, RU-C-F27*		

Tabla 123 - Requisito de capacidad RS-C-F59

RS-C-F60	
Título	Realización de movimientos *Controlador*

Descripción	Dependiendo de la petición realizada, las acciones que realizará el *Controlador* sobre el vehículo serán:
	• Peticiones cámara:
	  ○ Girar izquierda:
	  ▪ Moverá en sentido antihorario el servomotor encargado de realizar los movimientos horizontales de la cámara, hasta recibir una petición de cámara para parar de girar a la izquierda o llegar al límite de giro del servomotor.
	  ○ Girar derecha:
	  ▪ Moverá en sentido horario el servomotor encargado de realizar los movimientos horizontales de la cámara, hasta recibir una petición de cámara para parar de girar a la derecha o llegar al límite de giro del servomotor.
	  ○ Parar girar derecha o izquierda:
	  ▪ Detendrá el movimiento del servomotor encargado de realizar los movimientos horizontales de la cámara.
	  ○ Girar arriba:

        - Moverá en sentido antihorario el servomotor encargado de realizar los movimientos verticales de la cámara, hasta recibir una petición de cámara para parar de girar arriba o llegar al límite de giro del servomotor.
    - Girar abajo:
        - Moverá en sentido horario el servomotor encargado de realizar los movimientos verticales de la cámara, hasta recibir una petición de cámara para parar de girar abajo o llegar al límite de giro del servomotor.
    - Parar girar abajo o arriba:
        - Detendrá el movimiento del servomotor encargado de realizar los movimientos horizontales de la cámara.
    - Centrar:
        - Moverá los servomotores horizontal y vertical de la cámara al punto medio del recorrido de giro que puedan realizar.
- Peticiones vehículo:
    - Avanzar:
        - Moverá en sentido horario los dos motores que se encargan de la propulsión del vehículo, hasta recibir una petición de parar.
    - Retroceder:
        - Moverá en sentido antihorario los dos motores que se encargan de la propulsión del vehículo, hasta recibir una petición de parar.
    - Parar:
        - Detendrá el movimiento de los dos motores que se encargan de la propulsión.
    - Girar izquierda:
        - Moverá el motor encargado de la dirección en sentido antihorario y el motor izquierdo de propulsión en sentido horario. El motor de la dirección se moverá hasta recibir una petición de parar a girar a la izquierda o llegar al límite de giro del que dispone. El motor izquierdo de propulsión parará al recibir una petición de parar a girar a la izquierda.
    - Girar derecha:
        - Moverá el motor encargado de la dirección en sentido horario y el motor derecho de propulsión en sentido horario. El motor de la dirección se moverá hasta recibir una petición de parar a girar a la derecha o llegar al límite de giro del que dispone.

	El motor izquierdo de propulsión parará al recibir una petición de parar a girar a la izquierda. ○ Parar de girar izquierda o derecha: ▪ Detendrá el movimiento del motor encargado de la dirección del vehículo y los dos motores de propulsión. ○ Centrar dirección: ▪ Moverá el motor encargado de la dirección al punto medio del recorrido de giro que pueda realizar.		
	Alta	**Verificabilidad**	Baja
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-C-F26*		

Tabla 124 - Requisito de capacidad RS-C-F60

3.3.2. Requisitos de restricción

En este apartado se especificarán los distintos requisitos de software de restricción que el sistema debe de cumplir.

3.3.2.1. Requisitos de rendimiento

En este apartado se muestran los requisitos de rendimiento de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.3.2.1.1. Aplicación móvil

RS-R-P01			
Título	La pantalla SplashScreen se mostrará durante 3sg		
Descripción	La *aplicación móvil* utilizará una *Timertask* para mostrar durante 3sg la pantalla *SplashScreen*, pasados los 3sg, la aplicación móvil pasará a la pantalla *Principal.*		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Desechable	**Estabilidad**	No
RU relacionados	*RU-R-P01*		

Tabla 125 - Requisito de restricción RS-R-P01

3.3.2.2. Requisitos de seguridad

En este apartado se muestran los requisitos de seguridad de la aplicación divididos en tres apartados distintos, uno por cada componente de la aplicación.

3.3.2.2.1. Aplicación móvil

RS-R-S01			
Título	Error establecimiento conexión *socket* del *Servidor* en pantalla *ManualControl*		
Descripción	Si sucede un error al intentar establecer conexión con *socket* del *Servidor* en la pantalla *ManualControl*, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Notificará el error de conexión con el *Servidor* al usuario mediante una notificación de tipo *Toast*. • Volverá a la pantalla *Principal*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-S01*		

Tabla 126 - Requisito de restricción RS-R-S01

RS-R-S02			
Título	Error establecimiento conexión *socket* del *Servidor* en pantalla *ExecutingTray*		
Descripción	Si sucede un error al intentar establecer conexión con *socket* del *Servidor* en la pantalla *ExecutingTray*, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Notificará el error de conexión con el *Servidor* al usuario mediante una notificación de tipo *Toast*. • Volverá a la pantalla *ExecuteTray*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-S01*		

Tabla 127 - Requisito de restricción RS-R-S02

RS-R-S03			
Título	Error envío petición vehículo en pantalla *ManualControl*		
Descripción	Si en la pantalla *ManualControl* sucede un error por perdida de conexión con el *Servidor* al realizarle una petición, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Notificará el error de conexión con el *Servidor* al usuario mediante una notificación de tipo *Toast*. • Cerrará todas las conexiones con el *Servidor*. • Volverá a la pantalla *Principal*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-S02*		

Tabla 128 - Requisito de restricción RS-R-S03

RS-R-S04			
Título	Error envío petición vehículo en pantalla *ExecutingTray*		
Descripción	Si en la pantalla *ExecutingTray* sucede un error por perdida de conexión con el *Servidor* al realizarle una petición, la *Aplicación móvil* realizará las siguientes acciones en el orden que se citan: • Notificará el error de conexión con el *Servidor* al usuario mediante una notificación de tipo *Toast*. • Cerrará todas las conexiones con el *Servidor*. • Volverá a la pantalla *ExecuteTray*.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-S02*		

Tabla 129 - Requisito de restricción RS-R-S04

3.3.2.2.2. Servidor

RU-R-S05			
Título	Fallo de conexión con *Aplicación móvil*		
Descripción	Si sucede un error por perdida de conexión con el usuario de la *Aplicación móvil*, el *Servidor* realizará las siguientes acciones en el orden que se citan: • Finalizará la ejecución de las funciones *Control manual* o *Control automático*, si está ejecutando alguna de ellas. • Cerrará la comunicación con el usuario. • Volverá a esperar en el *socket* el establecimiento de la conexión con un usuario de la Aplicación móvil.		
Prioridad	Alta	**Verificabilidad**	Media
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-S03*		

Tabla 130 - Requisito de restricción RS-R-S05

3.3.2.3. Requisitos de implantación

En este apartado se muestran los requisitos de implantación de la aplicación divididos tres apartados distintos, uno por cada componente de la aplicación.

3.3.2.3.1. Aplicación móvil

RS-R-I01			
Título	API de Android mínima		
Descripción	El atributo *minSDKVersion* del *AndroidManifest.xml* tomará el valor 23, por lo que no se podrá instalar en dispositivos que dispongan de una versión de Android inferior a la 6.0.0 Mashmallow.		
Prioridad	Baja	**Verificabilidad**	Media
Necesidad	Desechable	**Estabilidad**	No
RU relacionados	*RU-R-I01*		

Tabla 131 - Requisito de restricción RS-R-I01

RS-R-I02			
Título	Conexión a la misma red que el *Servidor* para conexión a *socket*		
Descripción	Para realizar la conexión al *socket* del *Servidor*, la aplicación móvil deberá de ejecutarse sobre un dispositivo móvil conectado a la misma red que el *Servidor*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-I02*		

Tabla 132 - Requisito de restricción RS-R-I02

3.3.2.3.2. Servidor

RS-R-I03			
Título	Red de conexión necesaria para *socket* y transmisión de imágenes		
Descripción	El *Servidor* deberá de estar conectado a una red para poder crear el *socket* para la conexión de nuevos usuarios de la *aplicación móvil* y establecer conexiones con ellos. Además, esta red también es necesaria para que el *Servidor* pueda transmitir las imágenes a través de la dirección *http*.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-I03*		

Tabla 133 - Requisito de restricción RS-R-I03

RS-R-I04			
Título	Disponibilidad de una cámara para transmisión de imágenes y conducción autónoma.		
Descripción	Para poder llevar a cabo la transmisión de imágenes y la conducción autónoma, el *Servidor* deberá de ejecutarse sobre un hardware que incorpore o tenga conectada una cámara		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-I04*		

Tabla 134 - Requisito de restricción RS-R-I04

RS-R-I05			
Título	Conexión puerto serial *Controlador*		
Descripción	Para llevar acabo la comunicación con este a través del puerto serial, el *Servidor* deberá ejecutarse sobre un dispositivo que incorpore o tenga conectado el dispositivo que ejecuta el *Controlador.*		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-I05*		

Tabla 135 - Requisito de restricción RS-R-I05

3.3.2.3.3. Controlador

RS-R-I06			
Título	Fallo compilación y carga programa *Controlador*		
Descripción	Se producirá un error de compilación si se intenta compilar y cargar el programa *Controlardor* con un Arduino IDE sin las librerías del *Shield* Adafruit Motor/Stepper/Servo Shield v1.2.		
Prioridad	Alta	**Verificabilidad**	Alta
Necesidad	Esencial	**Estabilidad**	Si
RU relacionados	*RU-R-I06*		

Tabla 136 - Requisito de restricción RS-R-I06

3.4. Matrices de trazabilidad

Una vez se han definido los requisitos de usuario y se han extraído los requisitos de software a partir de ellos, se comprobará que todos los requisitos de usuarios están contemplados al menos en un requisito de software.

Para dejar reflejada esta comprobación se utilizan matrices de trazabilidad. El formato de las matrices será el siguiente:

	ID requisito de usuario 1	…	ID requisito de usuario n
ID requisito de software 1			
….			
ID requisito de software n			

Tabla 137 - Ejemplo matriz de trazabilidad

- **ID requisito de software X**: Indica el ID de un requisito de software.
- **ID requisito de usuario X:** Indica el ID de un requisito de usuario.

Si un requisito de software esta contemplado de forma completa o parcial por un requisito de usuario, se indicará mediante una "X" en la celda cuya fila corresponda con el ID del requisito de usuario y su columna con el ID del requisito de software.

Si cada columna tiene al menos una "X", significará que se han contemplado todos los requisitos de usuario en los requisitos de software. En caso contrario, se deberá de revisar el conjunto de requisitos de software, realizando las modificaciones que sean necesarias para que este conjunto contemple todos los requisitos de usuario.

Para facilitar la lectura de estas matrices de trazabilidad, se realizarán 3 matrices de trazabilidad. Cada una de estas matrices contemplará los requisitos de usuario y software de un componente de la aplicación.

3.4.1. Matriz de trazabilidad requisitos de Aplicación móvil

En este apartado se mostrará la matriz de trazabilidad entre los requisitos de usuario y de software definidos para el componente *Aplicación móvil.*

	RU-C-F01	RU-C-F02	RU-C-F03	RU-C-F04	RU-C-F05	RU-C-F06	RU-C-F07	RU-C-F08	RU-C-F09	RU-C-F10	RU-C-F11	RU-C-F12	RU-R-P01	RU-R-S01	RU-R-S02	RU-R-I03	RU-R-I04
RS-C-F01	X																
RS-C-F02		X															
RS-C-F03		X	X														
RS-C-F04		X		X													
RS-C-F05		X	X														
RS-C-F06			X														
RS-C-F07			X								X						
RS-C-F08			X														
RS-C-F09			X														
RS-C-F10				X													
RS-C-F11				X													
RS-C-F12				X		X											
RS-C-F13						X											
RS-C-F14						X											
RS-C-F15						X											
RS-C-F16					X	X											
RS-C-F17						X											
RS-C-F18							X										
RS-C-F19							X	X									
RS-C-F20							X	X									
RS-C-F21							X		X								
RS-C-F22							X			X							
RS-C-F23							X										
RS-C-F25									X								
RS-C-F26									X								
RS-C-F27					X				X								
RS-C-F28									X								
RS-C-F29								X									
RS-C-F30								X									
RS-C-F31								X									
RS-C-F32								X									
RS-C-F33								X			X	X					
RS-C-F34								X									
RS-C-F35		X		X		X	X	X									
RS-C-F36												X					
RS-C-F37			X					X				X					
RS-C-F38					X												
RS-C-F39					X												
RS-C-F40					X												
RS-R-P01													X				
RS-R-S01														X			
RS-R-S02														X			
RS-R-S03															X		
RS-R-S04															X		
RS-R-I01																X	
RS-R-I02																	X

Tabla 138 - Matriz de trazabilidad requisitos Aplicación móvil

3.4.2. Matriz de trazabilidad requisitos de Servidor

En este apartado se mostrará la matriz de trazabilidad entre los requisitos de usuario y de software definidos para el componente *Servidor.*

	RU-C-F13	RU-C-F14	RU-C-F15	RU-C-F16	RU-C-F17	RU-C-F18	RU-C-F19	RU-C-F20	RU-C-F21	RU-C-F22	RU-C-F23	RU-C-F24	RU-R-S03	RU-C-I03	RU-R-I04	RU-R-I05
RS-C-F41	X															
RS-C-F42		X														
RS-C-F43		X														
RS-C-F44		X														
RS-C-F45			X	X	X						X					
RS-C-F46				X		X										
RS-C-F47						X	X				X					
RS-C-F48						X	X				X					
RS-C-F49						X				x	X					
RS-C-F50						X					X					
RS-C-F51						X		X		X						
RS-C-F52				X			X	X			X					
RS-C-F53								X	X							
RS-C-F54								x		X	X					
RS-C-F55								x			X					
RS-C-F56												X				
RS-R-S05													X			
RS-C-I03														X		
RS-C-I04															X	
RS-C-I05																X

Tabla 139 - Matriz de trazabilidad requisitos Servidor

3.4.3. Matriz de trazabilidad requisitos Controlador

En este apartado se mostrará la matriz de trazabilidad entre los requisitos de usuario y de software definidos para el componente *Controlador.*

	RU-C-F25	RU-C-F26	RU-C-F27	RU-R-I06
RS-C-F57	X	X		
RS-C-F58		X	X	
RS-C-F59	X	X	X	
RS-C-F60		X		
RS-R-I06				X

Tabla 140 - Matriz de trazabilidad requisitos Controlador

4. Diseño

En este apartado se analizarán las decisiones de diseño tomadas en cuanto a el diseño arquitectónico del sistema que conforma la aplicación, las interfaces con las que interactúa el usuario, la estructura de la base de datos de la *Aplicación móvil* y las decisiones tomadas a la hora de realizar la implementación. Además, en el apartado de las decisiones tomadas en la implementación, se indicarán aquellas librerías y funcionalidades que utiliza el sistema, pero han sido desarrolladas por terceros, indicando el motivo de su elección.

4.1. Diseño de la arquitectura del sistema

La aplicación se basa en una arquitectura Cliente-Servidor. Este, es un tipo de arquitectura distribuida que asigna roles diferentes a los programas o procesos que se comunican. Los roles que se asignan son:

- **Cliente**: Es el rol que se le asigna a el programa que realiza peticiones a otro, el servidor. Este es el elemento activo de la comunicación.
- **Servidor:** Es el rol que se le asigna al programa que responde a las peticiones de los clientes. Este es el elemento pasivo de la comunicación.

En esta aplicación existen tres componentes principales que interactúan entre si: La *Aplicación móvil*, el *Servidor* y el *Controlador*. La *Aplicación móvil* realiza peticiones al *Servidor*, el cual las responde, y el *Servidor* realiza peticiones al *Controlador*, el cual las responde, pero, la *Aplicación móvil* nunca realiza peticiones de forma directa al *Controlador*. Por lo tanto, se podría considerar que tenemos 2 arquitecturas cliente-servidor, pero debido a que el *Servidor* nunca interactúa con el *Controlador* salvo que previamente le haya realizado una petición la *Aplicación móvil*, se utilizará para esta aplicación una arquitectura Cliente-Servidor de tres capas. En este caso, las capas de esta arquitectura son las siguiente:

- **Capa 1:** La *Aplicación móvil*, ya que es la que siempre actuará como cliente de *Servidor*. Además, la *Aplicación móvil* es la que ofrece la interfaz mediante la cual el cliente interactuará con el sistema que conforma la aplicación.
- **Capa 2:** El *Servidor*, debido a que será el que responda a las peticiones de la *Aplicación móvil*, y para llevar a cabo estas peticiones, deberá de realizar peticiones al *Controlador*, actuando de capa intermedia entre la *Aplicación móvil* y el *Controlador*.
- **Capa 3:** El *Controlador*, ya que es quien responde a las peticiones del *Servidor* para que este pueda llevar a cabo las peticiones de la *Aplicación móvil*.

En esta arquitectura se ha obviado la base de datos de la *Aplicación móvil*, debido a que esta base de datos solo es utilizada por la *Aplicación móvil* y el resto de componentes no interactúa con ella.

A continuación, se realiza una representación gráfica de la arquitectura Cliente-Servidor de 3 capas que se utiliza en este sistema:

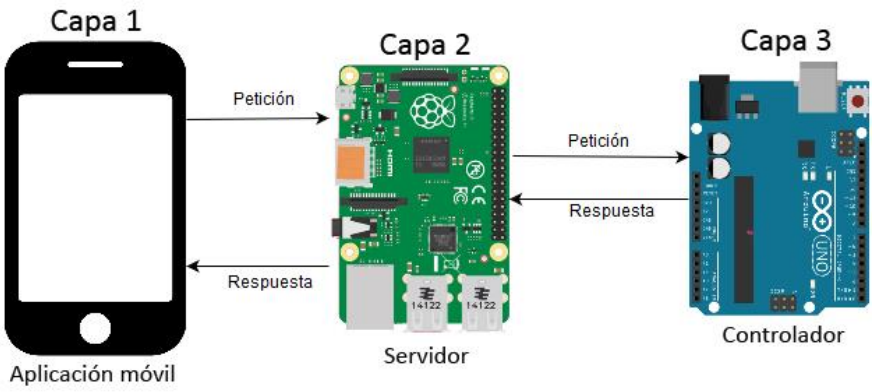

Ilustración 34 - Arquitectura Cliente-Servidor de la Aplicación

Para el desarrollo de esta arquitectura se ha seguido el patrón de diseño modelo-vista-controlador. En este patrón de diseño, se separan los datos de la aplicación y la lógica de negocio, de la interfaz del usuario. Los tres componentes principales de este patrón son los siguientes:

- **Modelo**: Este componente se encarga de representar la información con la que el sistema opera, además, se encarga de gestionar cualquier tarea relativa a la manipulación de esta información.
- **Vista**: Es el componente encargado de representar el modelo de forma visual y que el usuario pueda interactuar con él.
- **Controlador**: Es el componente que responde a las acciones de los usuarios. Este es el encargado de realizar peticiones al modelo y devolver la información obtenida a la vista.

A continuación, se muestra el flujo de funcionamiento de un sistema cuya arquitectura sigue el patrón de diseño modelo-vista-controlador [B57].

Ilustración 35 – Flujo de funcionamiento MVC

Una vez se conoce el tipo de arquitectura y el patrón de diseño seguido, se realiza la división de la aplicación en los distintos componentes que la forman.

Se conoce de antemano que el sistema está compuesto de 3 componentes principales, la *Aplicación móvil*, el *Servidor* y el *Controlador*, por lo que se realizará la división de cada uno de estos componentes en sus diferentes subcomponentes.

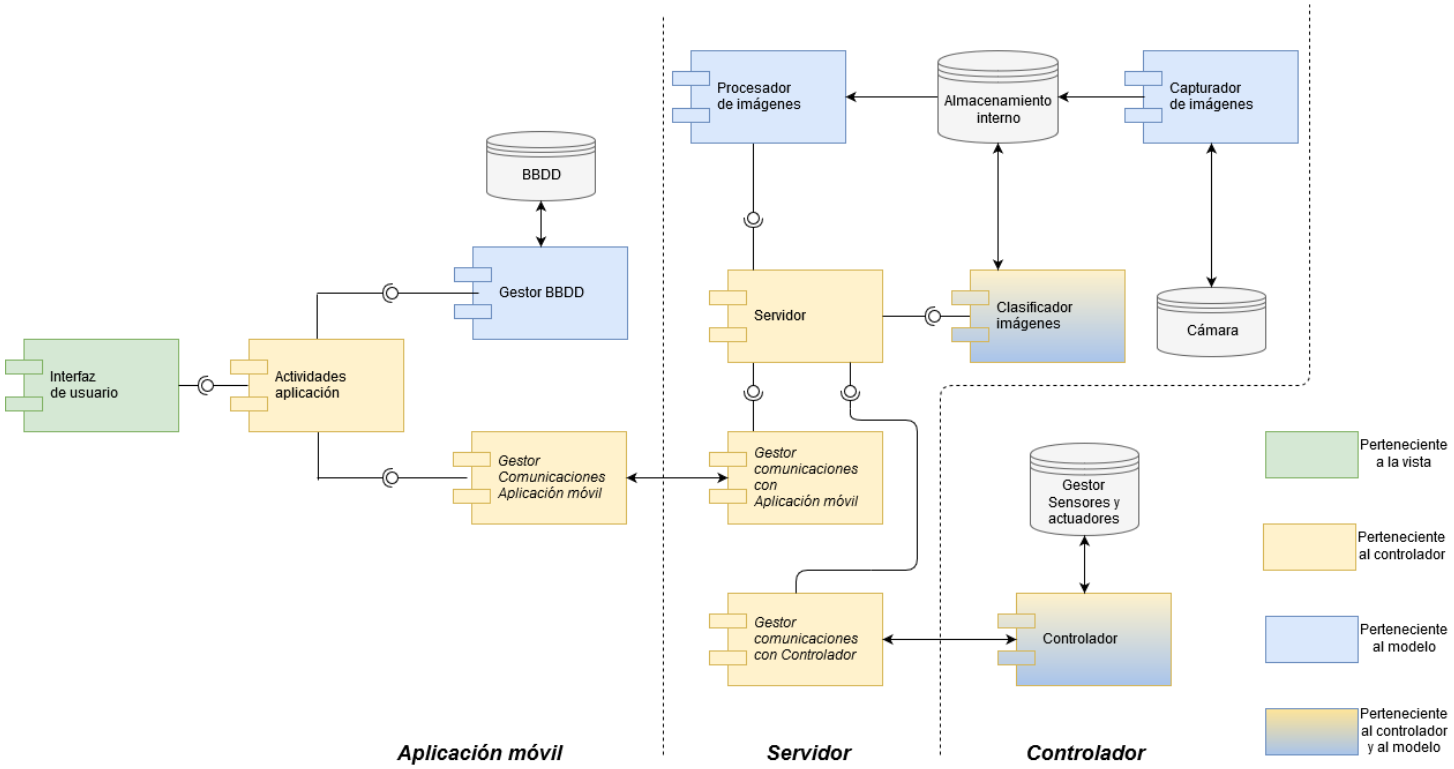

Ilustración 36 - División de Aplicación móvil, Servidor y Controlador en subcomponentes

Como se observa, aunque se ha seguido el patrón de diseño MVC, no se ha conseguido llevar a cabo de forma estricta, existiendo algunos componentes que pertenecen al controlador y al modelo. A continuación, se especifica cada uno de los subcomponentes en detalle. Para ello, se rellena por cada componente una tabla con el siguiente formato:

ID	
Nombre	
Componente	
Descripción	
Subcomponentes	
Dependencias	

Tabla 141- Formato tabla componentes de la aplicación

- **ID:** Identificador univoco del componente, cada componente tendrá su propio ID que lo identificará con respecto al resto de componentes. Este tendrá el siguiente formato:

  SCOM-XX

  - *SCOM:* Indica que es un subcomponente.
  - *XX:* Indica el número de subcomponente, la asignación de este número dependerá de su orden de creación.
- **Nombre:** Indica el nombre que se le ha asignado a dicho subcomponente.
- **Componente:** Indica el nombre que se le ha asignado al componente al que pertenece el subcomponente.
- **Descripción:** Descripción breve de aquello que realiza el subcomponente.
- **Funciones:** Indica el listado de funcionalidades que realiza dicho subcomponente.
- **Dependencias:** Indica el listado de los distintos subcomponentes que el subcomponente necesita de forma directa para funcionar correctamente.

SCOM_01	
Nombre	Interfaz de usuario
Componente	*Aplicación móvil*
Descripción	Subcomponente que representa la interfaz gráfica de la *Aplicación móvil* con la que el usuario va a interactuar con él sistema.
Funciones	• **Interactuación del usuario con el sistema:** Permite que el usuario pueda interactuar con el sistema que conforma la aplicación, a través de la *Aplicación móvil*.
Dependencias	• **Actividades aplicación**

Tabla 142 - Subcomponente SCOM_01

SCOM_02	
Nombre	Actividades aplicación
Componente	*Aplicación móvil*
Descripción	Subcomponente que representa a la lógica que hay detrás de todas las pantallas que conforman la *Aplicación móvil*.
Funciones	• **Funcionalidad en la interfaz gráfica:** Es el encargado de que cuando el usuario interactúe con algún elemento de la interfaz gráfica, la *Aplicación móvil* lleve a cabo la funcionalidad asignada a dicho elemento. • **Transición entre actividades:** Se encarga de llevar acabo la transición entre las pantallas de la aplicación. • **Conexión, envió de peticiones y recepción de respuestas del *servidor:*** Es el encargado de utilizar el *gesto de comunicaciones de Aplicación móvil* para establecer la conexión con el servidor, realizar el envío de peticiones cuando corresponda y recibir las respuestas del *Servidor*. • **Gestión de trayectos:** Mediante el uso del *Gestor BBDD*, lleva a cabo el almacenamiento, modificación y borrado de los trayectos almacenados en la BBDD de la *Aplicación móvil*.
Dependencias	• **Gestor trayectos** • **Gestor de comunicaciones *Aplicación móvil***.

Ilustración 37 - Subcomponente SCOM_02

SCOM_03	
Nombre	Gestor de comunicaciones *Aplicación móvil*
Componente	*Aplicación móvil*
Descripción	Subcomponente que ofrece una interfaz para él envió de peticiones y recepción de respuestas del *Servidor*.
Funciones	• **Establecimiento de conexión:** Permite el establecimiento de la conexión con el *Servidor*. • **Enviar petición y espera de respuesta:** Permite el envío de peticiones al *Servidor* y esperar de forma indefinida a la respuesta. • **Finalizar conexión:** Permite la finalización de la conexión con el *Servidor*.
Dependencias	-

Tabla 143 - Subcomponente SCOM_03

SCOM_04	
Nombre	Gestor BBDD
	Aplicación móvil
Descripción	Subcomponente que ofrece una interfaz para la consulta, almacenamiento, modificación y borrado, de los trayectos almacenados en la BBDD de la *Aplicación móvil* sin necesidad de interactuar de forma directa con el *Gestor BBDD*.
Funciones	• **Generar BBDD:** Se encarga de generar la base de datos si esta no está creada. • **Insertar trayecto:** Permite insertar un nuevo trayecto en la BBDD. • **Listar trayectos:** Permite listar todos los trayectos de la BBDD. • **Obtener trayecto:** Permite obtener un trayecto específico del cual se conoce el ID. • **Modificar trayecto:** Permite modificar un trayecto del que se conoce su ID.
Dependencias	• **Gestor BBDD**

Tabla 144 - Subcomponente SCOM_04

SCOM_05	
Nombre	Gestor de comunicaciones con *Aplicación móvil*
Componente	*Servidor*
Descripción	Subcomponente que ofrece una interfaz para la realización de comunicaciones con un usuario que esté utilizando la *Aplicación móvil.*
Funciones	• **Crear socket recepción conexiones:** Permite la creación de un socket para la recepción de nuevas conexiones de usuarios de la *Aplicación móvil*. • **Espera y establecimiento de conexión:** Permite la espera indefinida hasta que se produzca la conexión de un usuario al *socket* y establece una conexión específica para realizar la comunicación con dicho usuario. • **Enviar mensaje**: Permite enviar un mensaje a través de una conexión establecida con la *Aplicación móvil*. • **Recibir mensaje**: Permite recibir un mensaje a través de una conexión establecida con la *Aplicación móvil*.
Dependencias	-

Tabla 145 - Subcomponente SCOM_05

SCOM_05	
Nombre	Gestor de comunicaciones con *Controlador*
Componente	*Servidor*
Descripción	Componente que se encarga de ofrecer una interfaz para la comunicación con el *Controlador* a través del puerto serial.
Funciones	• **Abrir conexión**: Permite la apertura de una conexión con el *Controlador*. • **Enviar mensaje:** Permite enviar un mensaje a través de una conexión establecida con el *Controlador*. • **Recibir mensaje**: Permite recibir un mensaje a través de una conexión establecida con el *Controlador*
Dependencias	-

Tabla 146 - Subcomponente SCOM_05

SCOM_06	
Nombre	Gestor de imágenes
Componente	*Servidor*
Descripción	Componente que se encarga de ofrecer una interfaz para el tratamiento de las imágenes capturadas por la cámara conectada al dispositivo que ejecuta el *Servidor*.
Funciones	• **Obtener nombre imagen:** Permite la obtención del nombre de las imágenes capturadas en el segundo actual por el *Capturador de imágenes*. • **Conversión imagen**: Permite, dado el nombre de una imagen, su carga y conversión en un formato comprensible para el *Clasificador de imágenes*.
Dependencias	• **Gestor de almacenamiento interno**

Tabla 147 - Subcomponente SCOM_06

COM_07	
Nombre	Clasificador de imágenes
Componente	*Servidor*
Descripción	Componente que se encarga de ofrecer una interfaz para la creación y uso de un clasificador de imágenes.
Funciones	• **Creación del clasificador de imágenes no entrenado:** Permite la creación del clasificador de imágenes sin entrenar. • **Creación del clasificador de imágenes entrenado:** Permite la creación del clasificador utilizando un fichero con los datos de un clasificador ya entrenado. • **Clasificar imagen:** Permite la clasificación de una imagen que previamente haya sido transformada a el formato correspondiente mediante el *Gestor de imágenes.* • **Generar fichero de entrenamiento:** Permite la generación de un fichero con los datos del clasificador. • **Entrenar clasificador:** Permite entrenar el clasificador introduciendo un conjunto de imágenes procesadas mediante el *Gestor de imágenes.*
Dependencias	-

Tabla 148 - Subcomponente SCOM_07

SCOM_08	
Nombre	Capturador de imágenes
Componente	*Servidor*
Descripción	Componente que se encarga de realizar la captura, almacenamiento y transmisión de las imágenes a través de la cámara conectada al vehículo. Este componente se lanzará como un proceso independiente.
Funciones	• **Capturar imágenes:** Se encarga de la captura de las imágenes a través de la cámara conectada al vehículo. • **Almacenamiento imágenes:** Realiza el almacenamiento, en una ruta específica, de las imágenes capturadas por la cámara conectada al hardware sobre el que se ejecuta. Además, el nombre de estas imágenes corresponderá con la fecha del segundo en el que se realizaron. • **Transmisión de imágenes:** Transmite las imágenes capturadas en la dirección http formada (IP dispositivo que ejecuta el servidor) + Puerto especificado.
Dependencias	• **Gestor de almacenamiento interno** • **Cámara**

Tabla 149 - Subcomponente SCOM_08

COM_09	
Nombre	Servidor
Componente	*Servidor*
Descripción	Subcomponente que se encarga de utilizar el resto de componentes para ofrecer al usuario los modos de conducción automático y manual.
Funciones	• **Obtención IP:** Obtiene la dirección IP del dispositivo sobre el que se ejecuta y realiza impresión por pantalla. • **Tratamiento de peticiones y generación de respuestas *Aplicación móvil*:** Es el encargado del tratamiento de las peticiones recibidas desde la *Aplicación móvil* y la generación de las respuestas a dichas peticiones. Además, se encarga de utilizar *Gestor de comunicaciones con Aplicación móvil* para enviarlas y recibirlas. • **Generación de peticiones y tratamiento de respuestas del *Controlador*:** Genera las peticiones que se realizan al *Controlador*, ya sean propias o enviadas por el usuario utilizando la *Aplicación móvil*. Además, también se encarga de la recepción de las respuestas del *Controlador* y su reenvió a la *Aplicación móvil.* Para llevar a cabo esto, utiliza el *Gestor de comunicaciones con Controlador* y el *Gestor de comunicaciones con Aplicación móvil*. • **Conducción autónoma**: Es el encargado de la toma de decisiones durante la conducción autónoma, utilizando el *Capturador de imágenes*, el *Gestor de imágenes* y el *Clasificador de imágenes.* • **Activación/desactivación cámara**: Es el encargado de lanzar y finalizar el *Capturador de imágenes*.
Dependencias	• **Gestor de comunicaciones con *Aplicación móvil.*** • **Gestor de comunicaciones con *Controlador.*** • **Gestor de imágenes.** • **Clasificador de imágenes.** • **Capturador de imágenes.**

Tabla 150 - Subcomponente SCOM_09

SCOM_10	
Nombre	Controlador
Componente	*Controlador*
Descripción	Componente que se encarga de la recepción de peticiones de movimiento del *Servidor* y su realización sobre el vehículo.
Funciones	• **Comunicaciones:** Realiza la recepción de peticiones y envío de respuestas al *Servidor* a través del puerto serial. • **Tratamiento de peticiones:** Se encarga del tratamiento de las peticiones recibidas del *Servidor* y la generación de su respuesta. • **Controlar actuadores:** Lleva a cabo el control de los actuadores dependiendo de las peticiones recibidas, mediante el uso del hardware sobre el que se ejecuta. • **Obtiene datos de sensores:** Obtiene la información del potenciómetro que indica la posición de la dirección.
Dependencias	• **Gestor cámara**

Tabla 151 - Subcomponente SCOM_10

4.2. Diseño de la base de datos

En esta aplicación la base de datos que se utiliza es muy pequeña y solo dispone de una única tabla, ya que solo se utiliza para el almacenamiento de los trayectos. Además, esta base de datos no es distribuida, debido a que esta se genera en el dispositivo móvil en el cual se instale la *Aplicación móvil*, lo que provoca que cada usuario disponga de su propia base de datos de trayectos y que solo pueda ser modificada por dicho usuario.

El gestor de bases de datos utilizado ha sido **SQLite**, ya que viene integrado en el SO Android y se ha utilizado con anterioridad para proyectos con BBDD de dimensiones reducidas.

A continuación, se indicará el nombre de la única tabla de la aplicación y atributos de esta:

- **trayectos:** Cada tupla de esta tabla almacena la información de un trayecto para realizar su posterior ejecución. Los campos de esta tabla son los siguientes:
    - **ID:** Es la clave primaria de la tabla. Es un campo de tipo entero autoincremental, por lo tanto, su valor se asignará de forma automática por el gestor de la BBDD según haya sido su orden de inserción en la tabla.
    - **Num_movmientos:** Campo de tipo entero que no se le puede asignar el valor NULL. Este almacenará el número de movimientos que tiene el trayecto**.**
    - **Movimientos:** Campo de tipo cadena de texto que no puede tomar el valor NULL. Esta almacena los movimientos del trayecto. El formato de esta cadena es "a:b:c:d:e:……" donde cada letra será un movimiento. Los movimientos posibles serán:
        - **"MOV: IZQ\n".** Girar hacia la izquierda.
        - **"MOV: DER\n".** Girar hacia la derecha.
        - **"MOV: FRT\n"**. Continuar de frente.
        - **"MOV: STP\n"**. Parar.

      Por último, se asegura mediante código, en las pantallas de *EditTray* y *NewTray*, que este campo no tenga más de 999 movimientos.
    - **Observaciones**: Campo de tipo cadena de texto que puede tomar el valor NULL. Se asegura mediante código en las pantallas *EditTray* y *NewTray*, que se acepten los caracteres UNICODE y que el máximo número de caracteres sea 1000.

Por lo tanto, la representación gráfica del diseño de la BBDD, indicando los tipos de datos en el formato utilizado por SQLite, es el siguiente:

trayectos	
ID	INTEGER PRIMARY KEY AUTOINCREMENT
num_movmientos	INTEGER NOT NULL
movimientos	TEXT NOT NULL
observaciones	TEXT

Tabla 152 - Diseño BBDD

4.3. Diseño de interfaces

En este apartado se indicará como se ha diseñado la interfaz gráfica de la *Aplicación móvil* y se explicará el motivo de dicho diseño. El resto de componentes no se incluyen ya que no disponen de una interfaz gráfica, el más cercano a esto es el *Servidor*, que dispone de una interfaz de línea de comandos mediante la que realiza impresiones por pantalla del estado del mismo.

A continuación, se muestra un diagrama de flujo de la *Aplicación móvil*, este se utiliza para dar una visión global de cómo navegará el usuario entre las distintas pantallas que componen la *Aplicación móvil*. Las pantallas se han clasificado en 4 categorías:

- **Pantalla inicial:** Aquella pantalla que ve el usuario al iniciar la aplicación.
- **Pantalla intermedia:** Aquellas pantallas que se utiliza para dar paso a otras pantallas.
- **Pantalla funcional:** Aquellas pantallas que se utilizan para llevar a cabo alguna funcionalidad del sistema.
- **Pantalla funcional e intermedia:** Es clasificación que se le asigna a aquellas pantallas con características de pantalla intermedia y funcional.

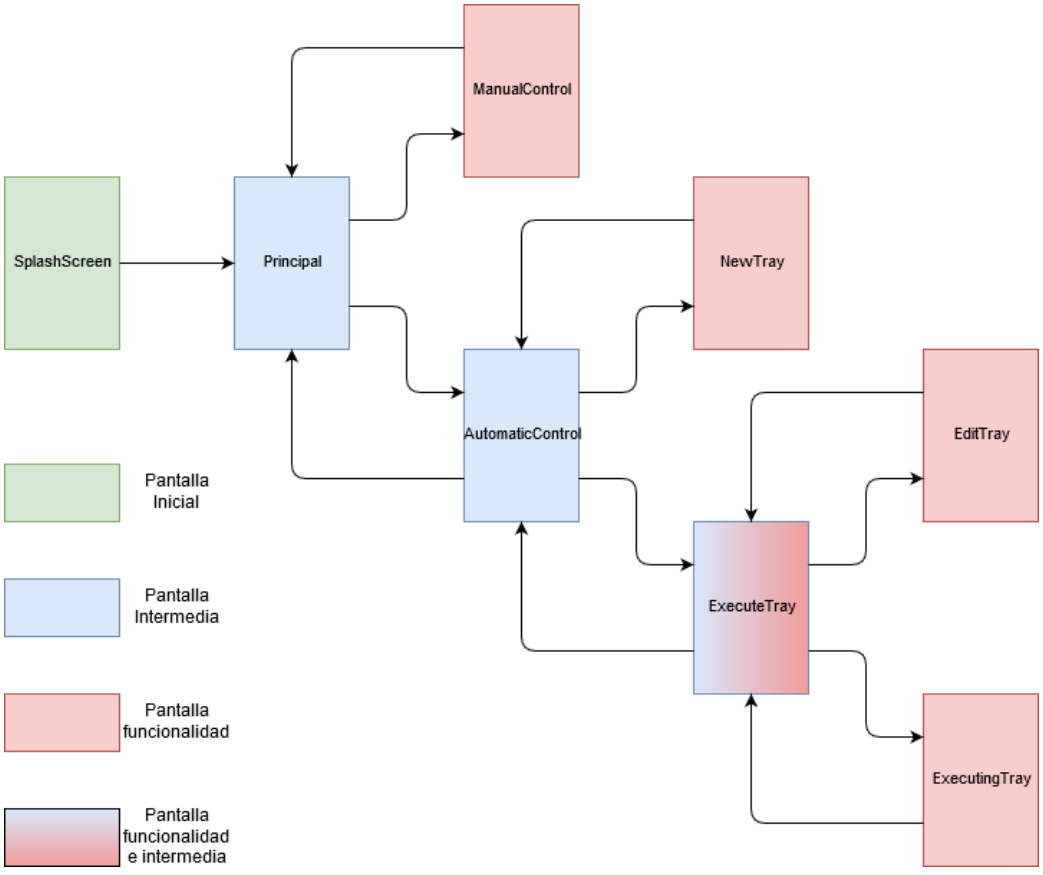

Ilustración 38 - Diagrama de flujo de la Aplicación móvil

A continuación, se realizará un análisis de la interfaz gráfica de cada una de las pantallas que componen la *Aplicación móvil*, justificando su diseño y comportamiento.

4.3.1. SplashScreen

Ilustración 39 - Pantalla SplashScreen de la Aplicación móvil

Es la primera pantalla que observa el usuario al inicializar la *Aplicación móvil*. Se utiliza como pantalla de presentación de la aplicación, la cual, pasada un tiempo dará paso de forma automática a la pantalla *Principal,* y no se podrá volver a esta pantalla salvo que se reinicie la *Aplicación móvil*.

Esta pantalla solo está compuesta de una imagen, el logotipo de la aplicación, centrada sobre un fondo azul.

Esta pantalla sigue estos principios de diseño:

- **Visibilidad del estado del sistema:** Esta pantalla informa al usuario de que se ha realizado la apertura de la *Aplicación móvil* y esta se está cargando.
- **Estética y diseño minimalista:** Ya que solo es el logotipo de la aplicación centrado sobre un fondo de color azul.
- **Prevención de errores:** Ya que el usuario no podrá realizar ninguna acción mientras se realiza la carga de la *Aplicación móvil*.
- **Se sigue la guía de diseño *Material Desing* de Google** [B44]**, respecto al apartado de pantallas de inicio** [B45]: En este apartado, se recomienda que las pantallas iniciales de la aplicación sean una interfaz vacía, que de la percepción carga, o una pantalla con la imagen de marca de la aplicación, que se muestre durante un breve periodo de tiempo mientras la aplicación está cargando. En este caso, se utiliza una pantalla con la imagen de marca de la aplicación, mostrando en el centro de la pantalla el logotipo de la aplicación. Debido a que esta aplicación es muy ligera, se mantiene esta pantalla de forma manual mediante código durante un mínimo de 3sg, ya que, si no se hiciera esto, la pantalla solo se mostraría durante unos breves instantes.

4.3.2. Principal

Ilustración 40 - Pantalla Principal de la Aplicación móvil

Pantalla que se muestra una vez cargada la aplicación. Es la pantalla de selección entre los dos apartados principales de la *Aplicación móvil*: *Control manual* y *Control automático*.

Esta pantalla dispone de dos botones de gran tamaño, uno encima del otro, que permiten seleccionar entre los dos apartados principales. Estos botones están compuestos de una imagen representativa del apartado al cual se accede mediante su pulsación y un texto que indica el nombre de dicho apartado.

Esta pantalla sigue estos principios de diseño:

- **Visibilidad del estado del sistema:** Esta pantalla informa de que la aplicación ya se ha cargado y se puede comenzar a utilizar.
- **Coincidencia entre el sistema y el mundo real:** Los botones de *Control manual* y *Control automático,* muestran imágenes que utilizan la correspondencia con el mundo real para ayudar al usuario a conocer que acciones podrá realizar con la *Aplicación móvil* si se pulsa alguno de dichos botones.
    - En el caso del botón de *Control manual*, se muestra una mano agarrando un volante, lo cual informa al usuario, que, al seleccionar el botón, se le llevará a las distintas opciones de control directo sobre el vehículo.
    - En el caso del botón de *Control automático*, se muestra una imagen de un mapa con un punto de inicio y otro de fin, lo cual informa al usuario, que, al seleccionar el botón, se le llevará a las distintas opciones de tratamiento de los trayectos.
- **Estética y diseño minimalista**: En esta pantalla se muestra una interfaz con únicamente dos botones, con el fin de organizar las distintas funciones de la

Aplicación móvil en dos apartados principales, e indicar que en dicha pantalla solo se puede realizar la selección entre esos dos apartados principales.

- **Reconocimiento antes que recuerdo:** En la *ActionBar* de esta pantalla se muestra en todo momento el nombre de la aplicación en la que se encuentra el usuario, en este caso, CRV que significa "Control Remoto de Vehículo".

Por otro lado, las acciones que realizan los botones de esta pantalla sobre la interfaz son las siguientes:

- **Botón *Control manual:*** Cuando el usuario pulse este botón se mostrará un cuadro de dialogo de tipo *AlertDialog* que se compone de los siguientes elementos:
  - **Título:** Indica que el cuadro de dialogo es para indicar la dirección IP del *Servidor* al cual se quiere conectar el usuario.
  - **Mensaje:** Indica el formato en el que se debería de indicar la dirección IP del *Servidor.*
  - **Campo de texto:** Dispone de un campo de texto para que el usuario pueda especificar la dirección IP *del Servidor.*
  - **Botón aceptar:** Permite indicar que la dirección IP del *Servidor* ya se ha especificado y dar paso a la pantalla *ManualControl*, y, por lo tanto, a la interfaz de dicha pantalla.
  - **Botón cancelar:** Permite el cierre del cuadro de dialogo y por lo tanto la cancelación de la especificación de la dirección IP del *Servidor*.

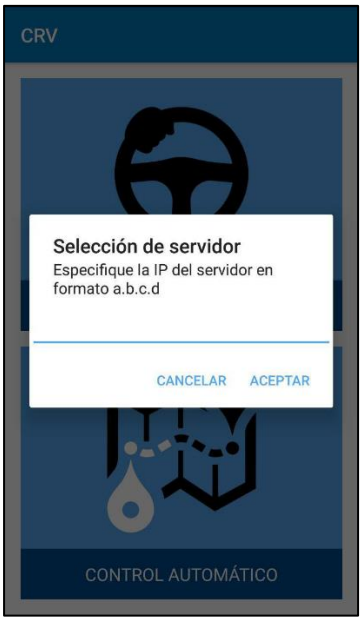

Ilustración 41 - Pantalla Principal de la aplicación móvil - Control manual - AlertDialog

Este cuadro de dialogo, también sigue una serie de principios de diseño:

- **Visibilidad del estado del sistema:** Ya que se informa al usuario de que va a conectarse a el *Servidor* de la aplicación desde la *Aplicación móvil*.
- **Control de usuario y libertad:** Ya que permite al usuario las siguientes acciones:

        - Indicar a que *Servidor* se quiere conectar.
        - Cerrar el cuadro de dialogo pulsando el botón *cancelar*.
        - Indicar cuando se realiza la conexión al *Servidor* y el paso a la pantalla *ManualControl* pulsando el botón *aceptar*.
    - **Prevención de errores:** Ofrece un botón cancelar, que permite al usuario cerrar el cuadro de diálogo si el botón de *control manual* fue pulsado por error o no se quiere establecer la conexión con el *Servidor*. Además, se indica el formato que debe de tener la dirección IP del *Servidor*, ayudando a que el usuario no tenga errores a la hora de indicar la IP.
    - **Reconocimiento antes que recuerdo:** Se le indica al usuario el formato que debe tener la IP sin necesidad de que este recuerde cual es.
- **Botón *Control automático:*** Cuando el usuario pulse este botón, se dará paso a la pantalla *AutomaticControl,* y, por lo tanto, dará paso a la interfaz de dicha pantalla.

Por último, si sucede algún error en la pantalla de *ManualControl* con la conexión del *Servidor*, ya sea por perdida de la conexión o imposibilidad de establecerla, y se produzca la vuelta a la pantalla *Principal*, esta mostrará una notificación de tipo *Toast* indicando el suceso. Esto no se incluye como parte del principio de diseño de *Visibilidad del estado del sistema*, ya que esta no se produce en esta pantalla sino en la pantalla *ManualControl* y se mantiene hasta esta pantalla.

Ilustración 42 - Pantalla Principal - Toast - Error servidor

4.3.3. ManualControl

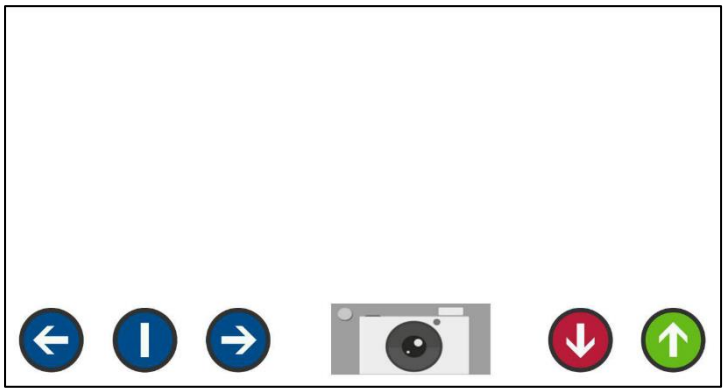

Ilustración 43 - Pantalla ManualControl - Cámara desactivada

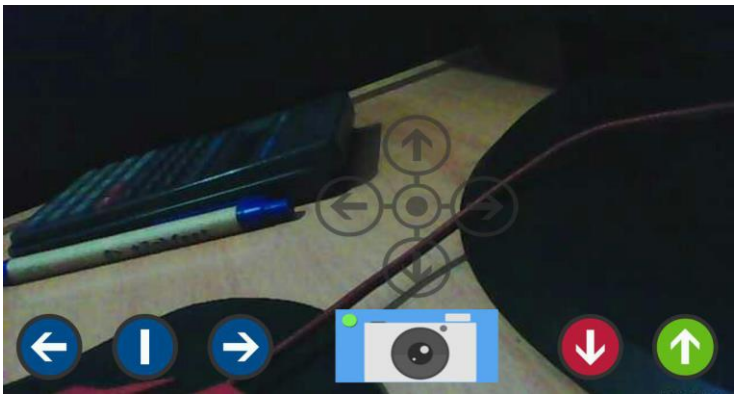

Ilustración 44 - Pantalla ManualControl - Cámara activa

Pantalla a la cual se accede a través de la pantalla *Principal*, si selecciona el botón de *Control manual* y se especifica de forma correcta la dirección IP del *Servidor* a conectarse. En esta pantalla es donde se realiza el *Control manual* del vehículo, indicando de forma directa todo aquel movimiento que se quiera que realice el vehículo. Esta pantalla se muestra en orientación horizontal, esto se debe a que para el control del vehículo es más cómoda que la vertical, además, las fotos tomadas por la cámara del vehículo para este modo son de 640x360, que es una resolución 16:9, al igual que la pantalla del dispositivo móvil cuando se orienta en horizontal.

Esta pantalla no dispone de *ActionBar*, ya que así se especificó en el requisito de sistema RS-C-F06. Esto permite que la pantalla disponga de una mayor cantidad de espacio para establecer su interfaz.

El *background* de la pantalla es un *WebView*, el cual permite acceder a páginas web dentro de una aplicación. Si la cámara está activa, este *WebView*, será el encargado de mostrar las imágenes capturadas por la cámara conectada al vehículo, las cuales son transmitidas a través de la dirección http:// (Dirección IP del Servidor):8001. Si la cámara no está activa, el WebView se ocultará, por lo que el *background* de la pantalla tomará su color por defecto, que es el blanco.

Por otro lado, esta pantalla dispone de una gran cantidad de botones, cada uno de ellos es encargado de llevar acabo alguna petición al *Servidor*. A continuación, se indicará que peticiones realiza cada uno de ellos al ser pulsados:

- **Botones con fondo azul**: De izquierda a derecha, estos botones se encargan de realizar las siguientes peticiones: Petición de giro del vehículo a la izquierda, petición de centrado de dirección y petición de giro del vehículo a la derecha.
- **Botón con fondo rojo:** Es el botón encargado de realizar las peticiones de movimiento al vehículo para que este retroceda.
- **Botón con fondo verde:** Es el botón encargado de realizar las peticiones de movimiento al vehículo para que este avance.
- **Botón con forma de cámara:** Es un botón de tipo *ToggleButton*. Cuando el tono de este botón está en escala de grises, indica que la cámara no ha sido activada, por lo que, al ser pulsado, manda una petición de activación de cámara y carga en el *WebView* la dirección http:// (Dirección IP del Servidor):8001. Cuando el botón no está en escala de grises, indica que la cámara está activa, por lo que, al ser pulsado, se envía una petición de desactivación de la cámara y se oculta el *WebView*.
- **Botones con fondo transparente:** Estos son los botones encargados de realizar las peticiones de movimiento de la cámara. Para especificar que petición realiza cada botón, se irán indicando los botones fila por fila:
    - **Primera fila:** El botón que hay en esta fila es el encargado de enviar la petición de giro de cámara hacia arriba.
    - **Segunda fila**: De izquierda a derecha, los botones de esta fila se encargan de enviar las siguientes peticiones: Petición de giro de cámara a la izquierda, petición de centrado de cámara, Petición de giro de cámara a la derecha.
    - **Tercera fila:** El botón que hay en esta fila, es el encargado de enviar la petición de giro de cámara hacia abajo.

Salvo los botones que realizan las peticiones de centrado de la cámara y el vehículo, todos estos botones, al dejar de ser pulsados, envían la petición que se encarga de finalizar el movimiento del cual realizaron la petición al ser pulsados.

Esta pantalla sigue estos principios de diseño:

- **Visibilidad del estado del sistema:** Permite observar lo que está realizando el vehículo a través de la cámara, además, mediante el *ToggleButton* que activa/desactiva la cámara, se puede conocer el estado de esta. Por otro lado, si la respuesta recibida por el *Servidor* indica que la petición no se ha podido realizar, o se pierde o no se puede establecer la conexión con este, se notificará mediante una notificación de tipo *Toast*.

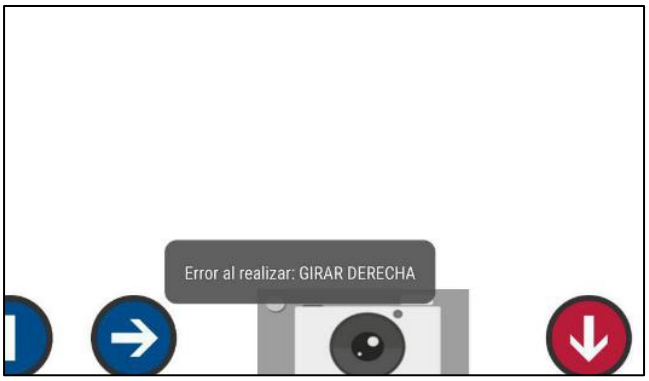

Ilustración 45 - Pantalla ManualControl - Cámara desactivada - Toast - Error petición

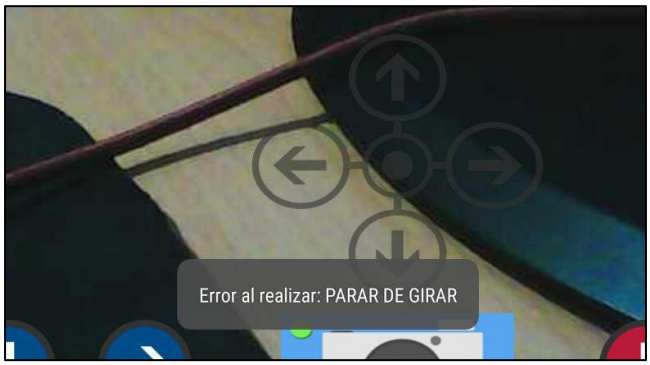

Ilustración 46 - Pantalla ManualControl - Cámara activa - Toast - Error petición

- **Coincidencia entre el sistema y el mundo real:** Todos los botones de la aplicación se han diseñado para que la interfaz sea lo más parecido a él mando de un videojuego, lo que le permitirá al usuario utilizar la interfaz de forma más fluida. Además, el botón de activación/desactivación de la cámara del vehículo, se ha representado con el icono de una cámara para ayudar al usuario a saber que este botón se utiliza para dicho propósito.
- **Prevención de errores:** Mediante el uso de un *ToggleButton* para la activación/desactivación de la cámara, se previene de que el usuario intente activar o desactivar la cámara cuando no se dan las condiciones para ello.
- **Reconocimiento antes que recuerdo:** El *ToggleButton* de la cámara permite observar de forma sencilla si la cámara está activa o no, sin necesidad de que el usuario tenga que recordarlo. Por último, debido a que podría haber confusiones entre los botones de control de los movimientos del vehículo y la cámara, los de la cámara solo se muestran cuando está activa, lo que ayuda a reconocerlos.

4.3.4. AutomaticControl

Ilustración 47 - Pantalla AutomaticControl

Pantalla a la cual se accede a través de la pantalla *Principal*, si selecciona el botón de *Control automático.* Es la pantalla de selección entre los dos subapartados principales del *Control automático*: *Nuevo trayecto* y *Ejecutar trayecto*. Se realiza la división en estos dos subapartados debido a que el de *Nuevo trayecto* se encarga de la creación de nuevos trayectos, mientras que el de *Ejecutar trayecto*, se encarga de las operaciones sobre los trayectos ya creados.

Al igual que ocurría con la pantalla *Principal*, esta pantalla dispone de dos botones de gran tamaño con las mismas características que los de la pantalla antes citada, uno encima del otro, pero en este caso, permiten seleccionar entre los dos subapartados del *Control automático*. Las acciones que realizan estos botones cuando son pulsadas por el usuario son las siguientes:

- **Botón *Nuevo Trayecto:*** Cuando el usuario pulse este botón, se pasará a la pantalla *NewTray,* dando paso a la interfaz de dicha pantalla.
- **Botón *Ejecutar Trayecto:*** Cuando el usuario pulse este botón, se pasará a la pantalla *ExecuteTray,* dando paso a la interfaz de dicha pantalla.

Esta pantalla sigue estos principios de diseño:

- **Coincidencia entre el sistema y el mundo real:** Los botones de *Nuevo Trayecto* y *Ejecutar trayecto,* muestran imágenes que utilizan la correspondencia con el mundo real, para ayudar al usuario a conocer que acciones podrá realizar con la *Aplicación móvil* si se pulsa alguno de dichos botones.
    - En el caso del botón de *Nuevo Trayecto*, se muestra un símbolo de adicción, que le indicará al usuario que es para añadir algún elemento, en este caso, un nuevo trayecto.

    - En el caso del botón de *Ejecutar trayecto*, es una imagen que muestra un móvil enviando un trayecto a un coche, lo que indicará al usuario que pulsando dicho botón podrá llevar acabo esta acción.
- **Estética y diseño minimalista**: Sigue la misma estética minimalista que la pantalla *Principal*, cuyo objetivo es agrupar las funciones del *Control automático* en dos subapartados, e indicar que en esta pantalla solo se puede seleccionar uno de estos dos subapartados o volver a la pantalla *Principal* mediante el *button back* de la *ActionBar*.
- **Reconocimiento antes que recuerdo:** En la *ActionBar* de esta pantalla, se muestra en todo momento el nombre del apartado de la aplicación en el que se encuentra el usuario, además, mediante el *button back* se muestra que hay una pantalla a la cual se puede volver.

Por último, si sucede el almacenamiento de un nuevo trayecto de forma correcta en la pantalla de *NewTray*, y se produzca la vuelta a la pantalla *AutomaticControl*, esta mostrará una notificación de tipo *Toast* indicando el suceso. Esto no se incluye como parte del principio de diseño de *Visibilidad del estado del sistema*, ya que esta no se produce en esta pantalla sino en la pantalla *NewTray* y se mantiene hasta esta pantalla.

Ilustración 48 - Pantalla AutomaticControl - Toast - Nuevo trayecto

4.3.5. NewTray

Ilustración 50 - Pantalla NewTray - Vacía

Ilustración 49 - Pantalla NewTray - Con movimientos

Pantalla a la cual se accede a través de la pantalla *AutomaticControl*, si selecciona el botón de *Nuevo trayecto.* Es la pantalla utilizada para la creación de nuevos trayectos, pudiendo indicando los movimientos y observaciones de cada nuevo trayecto.

Esta pantalla está dividida en tres partes principales. A continuación, se especifican los componentes de cada una de estas partes con los que puede interactuar el usuario:

1. **ActionBar:** Es la barra superior de la pantalla, dispone de dos botones:
    - ***Button back:*** Cuando el usuario pulse este botón, se volverá a la pantalla *AutomaticControl*.
    - **Botón confirmar:** Cuando el usuario pulse este botón, se mostrará un cuadro de dialogo de tipo *AlertDialog* que se compone de los siguientes elementos:
        - **Título:** Indica que el cuadro de dialogo es para indicar las observaciones del trayecto.
        - **Mensaje:** Indica que indica que no es estrictamente necesario indicar observaciones.
        - **Campo de texto:** Dispone de un campo de texto para que el usuario pueda especificar las observaciones del trayecto.
        - **Botón *aceptar*:** Al ser pulsado, se confirma la creación del trayecto, y, por lo tanto, que sus movimientos y observaciones están correctamente especificados.
        - **Botón *cancelar*:** Al ser pulsado, se produce el cierre del cuadro de dialogo y por lo tanto la cancelación del almacenamiento.

Ilustración 51 - Pantalla NewTray - Confirmar - AlertDialog

Este cuadro de dialogo, también sigue una serie de principios de diseño:

- **Visibilidad del estado del sistema:** Ya que se indica al usuario que se van especificar las observaciones del trayecto antes de almacenarse.
- **Control de usuario y libertad:** Ya que permite al usuario las siguientes acciones:
  - Indicar las observaciones del trayecto.
  - Cerrar el cuadro de dialogo pulsando el botón *cancelar*.
  - Indicar cuando se realiza el almacenamiento del trayecto pulsando el botón *aceptar*.
- **Prevención de errores:** Ofrece un botón cancelar, que permite al usuario cerrar el cuadro de diálogo si el botón de *confirmar* fue pulsado por error o se quiere modificar los movimientos del trayecto.

2. **Especificador de movimientos:** Es la mitad superior de la pantalla sin incluir el *ActionBar.* Dispone de tres filas de botones. A continuación, se especificará cada una de ellas de arriba abajo:
   - **Primera fila:** El botón de esta fila, al ser pulsado, añade el movimiento continuar de frente.
   - **Segunda fila:** El botón de la izquierda, al ser pulsado, añade el movimiento girar a la izquierda, mientras que el botón de la derecha, al ser pulsado, añade el movimiento girar hacia la derecha.
   - **Tercera fila:** El botón de esta fila, al ser pulsado, añade el movimiento parar.
3. **Listado de movimientos:** Es la mitad inferior de la pantalla. Este listado se encuentra inicialmente vacío, y es donde añaden los movimientos especificados mediante el uso de los botones del *Especificador de movimientos*. Estos movimientos se añaden al listado de arriba a abajo, siendo, el primer

movimiento del listado, el primero de los movimientos especificados. Cada uno de los elementos de este listado está compuesto por:

- **Una imagen representativa del movimiento.**
- **El nombre del movimiento.**
- **Botón borrar:** Este botón, al ser pulsado, elimina el movimiento del listado de movimientos, desplazando los movimientos por debajo de este, hacia arriba.

Esta pantalla sigue estos principios de diseño:

- **Visibilidad del estado del sistema:** Se utilizan notificaciones de tipo *Toast* para indicar la creación de forma correcta de un nuevo trayecto en la BBDD, e informar al usuario de que un trayecto debe tener al menos un movimiento y que no puede tener más de 999 movimientos.

Ilustración 52 - Pantalla NewTray - Toast - Al menos un movimiento

Ilustración 53 - Pantalla NewTray - Toast - Maximo 999 movimientos

- **Coincidencia entre el sistema y el mundo real:** El *especificador de movimientos*, al igual que los botones de la pantalla *ManualControl*, se ha diseñado para que se parezca lo máximo posible a un mando, elemento familiar para el usuario, que le permitirá usar la interfaz de forma más fluida. Como imagen representativa de cada movimiento, en el *listado de movimientos*, se han utilizado flechas que indican la dirección del movimiento. Por último, el botón borrar de cada movimiento tiene forma de "X" que es indicativo de cierre o borrado de un elemento.
- **Reconocimiento antes que recuerdo:** Mediante el listado de movimientos el usuario no tiene que recordar cuales son los movimientos que tiene el trayecto. Por otro lado, en la *ActionBar* se le indica al usuario el subapartado del *Control automático* en el cual se encuentra, y mediante el *button back sabe* que hay una pantalla anterior a la que puede volver, sabiendo entonces que esta no es la pantalla inicial de la *Aplicación móvil*.

4.3.6. ExecuteTray

Ilustración 54 - Pantalla ExecuteTray - Sin ningún trayecto

Ilustración 55 - Pantalla ExecuteTray - Con trayectos

Pantalla a la cual se accede a través de la pantalla *AutomaticControl*, si se selecciona el botón de *Ejecutar trayecto.* Es la pantalla utilizada para listar todos los trayectos almacenados en la BBDD de la Aplicación móvil, así como dar la posibilidad de realizar, editar y borrar cada trayecto del listado.

Esta pantalla tiene dos estados. A continuación, se especifican las características de cada estado:

- **Pantalla ExecuteTray vacía:** Si no se dispone de ningún trayecto en la BBDD, la pantalla *ExecuteTray* solo estará compuesta por un texto que aparece en mitad de dicha pantalla. El texto de esta pantalla informará al usuario de que no existen trayectos, e indicará a que pantalla se debe de dirigir para crear un nuevo trayecto.
- **Pantalla ExecuteTray con trayectos:** Si existen trayectos en la BBDD, se mostrarán mediante un listado. En este listado existen 4 campos, cuyas características son las siguientes:
    - **ID:** Indicará el ID de cada uno de los trayectos.
    - **NºMovimientos:** Indicará el número de movimientos de cada trayecto.
    - **Observaciones:** Indicará las observaciones indicadas por el usuario en cada trayecto.

- **Opciones:** Mostrará por cada trayecto 3 botones, estos botones tienen las siguientes características:
  - **Botón *Relizar trayecto***: Tiene el prácticamente el mismo comportamiento y principios de diseño que el botón *Control manual* de la pantalla *Principal* (4.3.2 PRINCIPAL). La única diferencia que existe, es en el botón *aceptar* del *AlertDialog*, por lo que a continuación se definirá su comportamiento:
    - **Botón aceptar:** Permite indicar que la dirección IP del *Servidor* ya se ha especificado, confirmar la selección del trayecto y dar paso a la pantalla *ExecutingTray*.

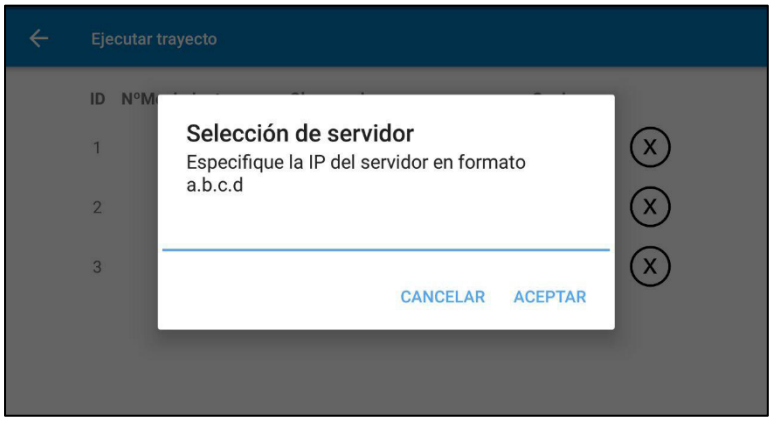

Ilustración 56 - Pantalla ExecuteTray - Realiza trayecto -AlertDialog

  - **Botón *Editar trayecto***: Cuando el usuario pulse este botón, se dará paso a la pantalla *EditTray* con los datos del trayecto del cual se pulso su botón *Editar trayecto*.
  - **Botón *Borrar Trayecto***: Cuando el usuario pulse este botón, se mostrará un cuadro de dialogo de tipo *AlertDialog* que se compone de los siguientes elementos:
    - **Título:** Indica que el cuadro de dialogo es para confirmar el borrado de un trayecto.
    - **Mensaje:** Pregunta al usuario si está seguro de querer borrar el trayecto seleccionado, indicando el ID de este.
    - **Botón *aceptar*:** Al ser pulsado, se confirma el borrado del trayecto, así como de todo su contenido.
    - **Botón *cancelar*:** Al ser pulsado, se produce el cierre del cuadro de dialogo y por lo tanto la cancelación del borrado.

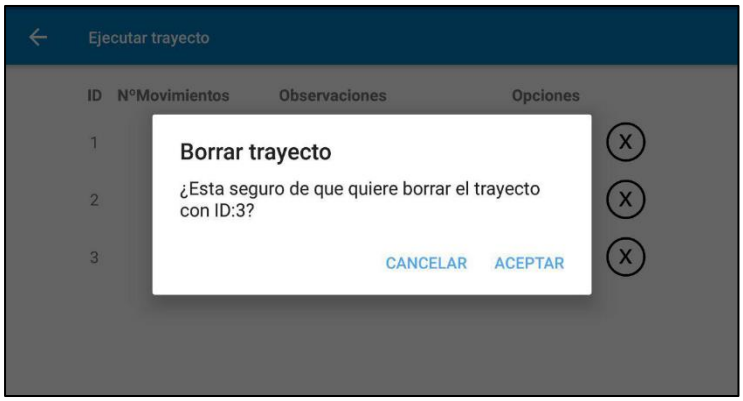

Ilustración 57 - Pantalla ExecuteTray - Borrar trayecto – AlertDialog

Este cuadro de dialogo, también sigue una serie de principios de diseño:

- **Visibilidad del estado del sistema:** Ya que se indica al usuario que se va realizar el borrado del trayecto.
- **Control de usuario y libertad:** Ya que permite al usuario las siguientes acciones:
  - Confirmar el borrado del trayecto pulsando el botón *aceptar*.
  - Cerrar el cuadro de dialogo pulsando el botón *cancelar*, por lo tanto, la cancelación del borrado.
- **Prevención de errores:** Ofrece un botón cancelar, que permite al usuario cerrar el cuadro de diálogo si el botón de *Borrar trayecto* fue pulsado por error.

Esta pantalla sigue estos principios de diseño:

- **Visibilidad del estado del sistema:** Se utilizan notificaciones de tipo *Toast* para indicar el borrado de forma correcta de un trayecto de la BBDD. Además, como ya se ha indicado antes, si la BBDD no tiene ningún trayecto, muestra en la interfaz un texto indicando esto y como solucionarlo.

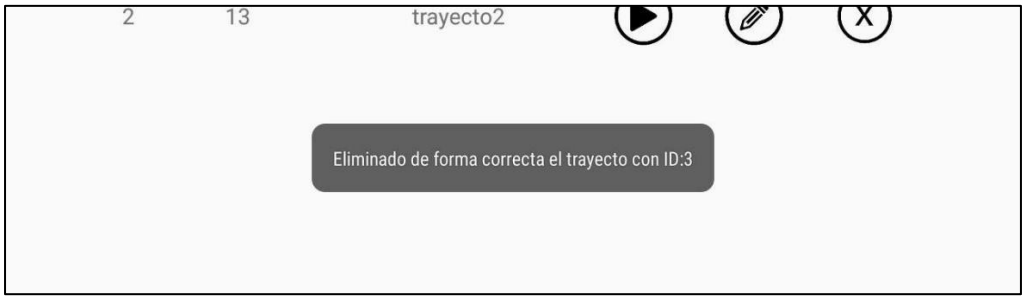

Ilustración 58 - Pantalla ExecuteTray - Toast - Trayecto eliminado

- **Coincidencia entre el sistema y el mundo real:** Los botones del campo de opciones del listado de trayectos, se han seleccionado para que el usuario sea capaz de saber que va a realizar la *Aplicación móvil* al pulsarlos, sin la necesidad de haber utilizado la aplicación con anterioridad. Por ello, el botón de *Realizar*

trayecto tiene la forma de un *Play* de un reproductor de música, el botón Editar trayecto es un lápiz, por último, el botón *Borrar trayecto* es una "X", que como ya se dijo en el apartado dedicado a la pantalla *NewTray*, este significa el cierre o borrado de un elemento.

- **Reconocimiento antes que recuerdo:** Mediante este listado de trayectos, el usuario no tiene que recordad el ID exacto de los trayectos para realizarlos, editarlos o borrarlos. Por último, como ya se ha comentado en anteriores pantallas, el *ActionBar* permite al usuario saber en que subapartado del *Control automático* se encuentra y conocer, mediante el *button back*, que esta no es la pantalla inicial de la aplicación, ya que se puede volver a una pantalla anterior.

Por último, si sucede algún error en la pantalla de *ExecutingTray* con la conexión del *Servidor*, ya sea por perdida de la conexión o imposibilidad de establecerla, y se produzca la vuelta a la pantalla *ExecuteTray*, esta mostrará una notificación de tipo *Toast* indicando el suceso. Esto también sucede si, se realiza la modificación de un trayecto de forma correcta en la pantalla de *EditTray*, y se produzca la vuelta a la pantalla *ExecuteTray*. Estos dos casos no se incluyen como parte del principio de diseño de *Visibilidad del estado del sistema*, ya que estas notificaciones no se producen en esta pantalla, sino en las pantallas *ExecutingTray* y *EditTray*, pero se mantienen hasta esta pantalla.

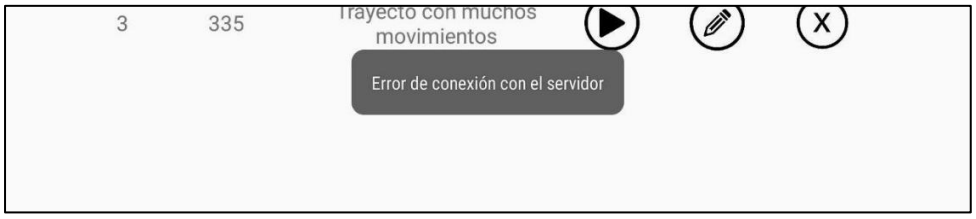

Ilustración 59 - Pantalla ExecuteTray - Toast - Error servidor

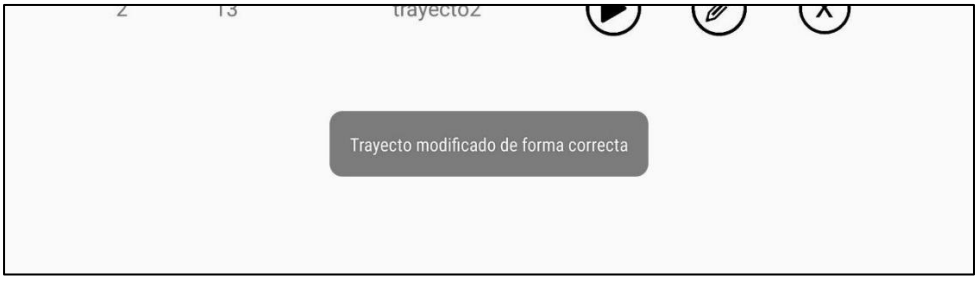

Ilustración 60 - Pantalla ExecuteTray - Toast - Trayecto modificado

4.3.7. EditTray

Ilustración 61 - Pantalla EditTray

Pantalla a la cual se accede a través de la pantalla *ExecuteTray*, si selecciona el botón de *Editar trayecto* de alguno de los elementos del listado de trayectos de dicha pantalla. Es la pantalla utilizada para la modificación de los trayectos almacenados en la BBDD, pudiendo añadir nuevos movimientos y eliminar los existentes, así como, modificar las observaciones.

Esta pantalla tiene prácticamente el mismo comportamiento y cumple los mismos principios de diseño que la pantalla *NewTray* (4.3.5 NEWTRAY) . Con unas pequeñas modificaciones:

- ***ActionBar***: este elemento muestra el texto Editar trayecto, en vez de, Nuevo trayecto.
- **Botón confirmar:** Al pulsar este botón, el AlertDialog generado es ligeramente diferente, los cambios de este son los siguientes:
    - **Mensaje:** Indica que indica que se pueden realizar cambios sobre las modificaciones, pero que esto no es estrictamente necesario.
    - **Campo de texto:** Tiene introducidas de antemano las observaciones del trayecto, para que el usuario pueda modificarlas.
    - **Botón *aceptar*:** Al ser pulsado, se confirma la modificación del trayecto, y, por lo tanto, que sus movimientos y observaciones, están correctamente especificados.
    - **Botón** *cancelar*: Al ser pulsado, se produce el cierre del cuadro de dialogo y por lo tanto la cancelación de la modificación.

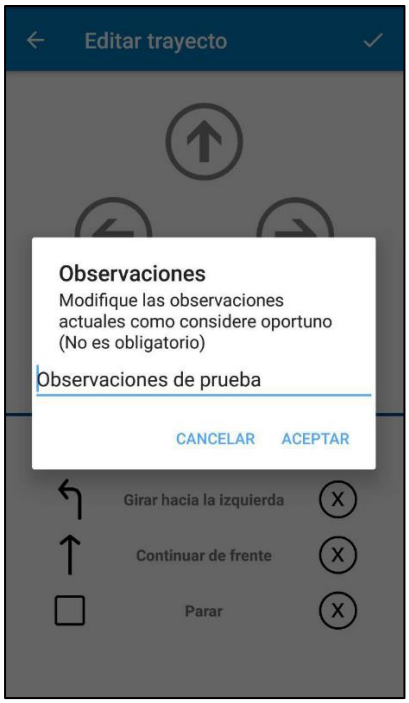

Ilustración 62 - Pantalla EditTray - Confirmar - AlertDialog

- **Listado de movimientos:** El listado de movimientos no se encuentra inicialmente vacío, sino que se encuentran de antemano los movimientos del trayecto a modificar.
- **Notificación de tipo *Toast* de modificación de trayecto:** Se cambia la notificación de tipo *Toast* que indicaba la creación de un nuevo trayecto *NewTray*, por una que indica la modificación correcta del trayecto.

Por último, esta pantalla también dispone de las notificaciones de tipo *Toast* para indicar que los trayectos deben de disponer de al menos un movimiento y pueden tener un máximo de 999 movimientos.

4.3.8. ExecutingTray

Ilustración 63 - Pantalla ExecutingTray

Pantalla a la cual se accede a través de la pantalla *ExecuteTray*, si selecciona el botón de *Realizar trayecto* de alguno de los elementos del listado de trayectos de dicha pantalla. Es la pantalla utilizada para el envío de trayectos al *Servidor* y observar la realización de dicho trayecto de forma autónoma por el vehículo, pudiendo observar las imágenes capturadas por la cámara del vehículo y el estado de realización de los movimientos del trayecto.

Esta pantalla está dividida en tres partes principales. A continuación, especificará las características de cada una de ellas:

1. ***ActionBar*:** Es la barra superior de la pantalla, dispone de:
    - **Texto:** Indica al usuario que se está realizando la ejecución de un trayecto.
    - ***Button back:*** Cuando el usuario pulse este botón, se volverá a la pantalla *ExecuteTray*.
2. **Visor de imágenes:** Es la mitad superior de la pantalla sin incluir el *ActionBar.* Al igual que el *background* de la pantalla *Principal,* es un *WebView*. Debido a que en esta pantalla siempre estará activa la cámara, se mostrarán constantemente las imágenes capturadas por esta en el *WebView*, para ello, el *WebView* accederá a la dirección http:// (Dirección IP del Servidor):8001 a la cual se transmiten las imágenes.

3. **Listado de movimientos:** Es la mitad inferior de la pantalla. En este listado se mostrarán los movimientos del trayecto que se está realizando. Cada uno de los movimientos de este listado dispondrán de los siguientes elementos:
    - **Una imagen representativa del movimiento.**
    - **El nombre del movimiento.**
    - **Indicador del estado de realización del movimiento:** Cada movimiento dispondrá de un indicador de su estado de realización. Significando cada uno de ellos lo siguiente:
        - ✓ **sobre fondo verde:** Se ha realizado de forma correcta.
        - ✕ **sobre fondo rojo:** No se ha podido llevar a cabo.
        - **? sobre fondo negro:** No se ha reconocido la respuesta del *Servidor* para dicho movimiento.
        - ↺ **sobre fondo azul:** Se está llevando a cabo.
        - **Sin imagen:** Todavía no se ha comenzado a realizar.

- **Visibilidad del estado del sistema:** Permite observar lo que está realizando el vehículo a través de la cámara, además, indica el momento en el momento en el que se realizaron las imágenes, mostrando la fecha de estas en el *Visor de imágenes*. Por otro lado, si se pierde o no se puede establecer la conexión con el *Servidor*, se notificará mediante una notificación de tipo *Toast*. Por último, se indica el estado de realización de cada movimiento en el *Listado de movimientos*.
- **Coincidencia entre el sistema y el mundo real:** El *listado de movimientos*, al igual que las pantallas *EditTray* y *NewTray*, utiliza como imagen representativa de los movimientos, flechas que indican la dirección del movimiento. Además, los colores e imágenes de los indicadores de estado de los movimientos han sido escogidos para facilitar al usuario el reconocimiento de dichos estados.
- **Reconocimiento antes que recuerdo:** Mediante el listado de movimientos el usuario no tiene que recordar cuales son los movimientos que tiene el trayecto. Por último, la *ActionBar* indica al usuario que acción está realizando la pantalla, y mediante el Button back de esta, se indica al usuario que existe una pantalla anterior a la que se puede volver.

4.4. Implementación

En este apartado, se indicará la implementación que se ha llevado a cabo en cada uno de los componentes principales de la aplicación, para ello, se explicará la implementación de cada uno de los subcomponentes que forman a estos componentes principales. Además, si alguno de los subcomponentes utiliza una librería externa o una funcionalidad desarrollada por terceros, se indicará, acompañada de los motivos de su elección.

En este apartado, se utilizará la identificación de componentes y subcomponentes realizada en el apartado 4.1 DISEÑO DE LA ARQUITECTURA DEL SISTEMA, específicamente en la ILUSTRACIÓN 36 - DIVISIÓN DE APLICACIÓN MÓVIL, SERVIDOR Y CONTROLADOR EN SUBCOMPONENTES.

4.4.1. Aplicación móvil

En este apartado, se hablará de la implementación realizada para el componente *Aplicación móvil*, el cual, como ya se ha comentado con anterioridad, es una aplicación para dispositivos móviles con el SO Android, la cual, utilizará el usuario para interactuar con el resto de componentes de la Aplicación.

A continuación, se especificará como se implementado cada uno de sus subcomponentes: *Gestor BBDD*, *Gestor de comunicaciones Aplicación móvil* y *Actividades aplicación*.

4.4.1.1. Gestor de BBDD

Este subcomponente, es el encargado de interactuar con la base de datos y dar una interfaz sencilla al resto de los subcomponentes de la *Aplicación móvil*, para que puedan interactuar con dicha BBDD sin necesidad de conocer ningún detalle sobre esta. Este componente se encuentra almacenado en un paquete denominado *data*, el cual está compuesto por dos clases: La clase *tray* y la clase *trayDB*. A continuación, la implementación de dichas clases.

4.4.1.1.1. Tray

Esta clase, se utiliza para representar mediante un objeto, una tupla de la tabla *trayectos* de la BBDD. Las instanciaciones de esta clase se utilizan para manejar y representar de una forma sencilla la información de los trayectos, sin tener que estar constantemente realizando sentencias *SQLite* contra la BBDD.

Esta clase, dispone de tantos atributos como la tabla *trayectos* de la BBDD, pero convertidos a los tipos de datos manejados por *Java*. Estos atributos serán privados.

Por último, esta clase tiene las funciones típicas de una clase utilizada para representar un objeto, disponiendo de: Un constructor vacío, un constructor donde se pueden especificar todos los atributos de la clase, *getters* y *setters* de los atributos, y las funciones *equals*, *hashcode* y *toString*.

4.4.1.1.2. TrayDB

Esta clase, se utiliza para que el resto de subcomponentes puedan interactuar con la BBDD sin necesidad de lanzar sentencias de *SQLite*, ya que esta clase se encargará de ello. La instanciación de esta clase abrirá una conexión para interactuar con la base de datos *Trayectos,* cuya única tabla se denomina *trayectos*, la cual ya se definió en TABLA 152 - DISEÑO BBDD. Esta clase heredará de la clase *SQLiteOpenHelper* [B46], que es una clase que ofrece Android para la creación de manejadores de base de datos *SQLite*, ofreciendo funciones todas las funciones necesarias para el tratamiento de este tipo de bases de datos.

Esta clase dispone de los siguientes atributos:

- **DATABASE_VERSION**: Atributo de tipo *int*. Especificará la versión de la base de datos.
- **DATABASE_NAME:** Atributo de tipo *String. A*lmacenará el nombre de la BBDD a utilizar, en este caso, la BBDD *Trayectos*.
- **TABLE_NAME:** Atributo de tipo *String*. Almacenará el nombre de la tabla de la BBDD *Trayectos* que se utiliza, en este caso, la tabla *trayectos*.
- **COLUMN_ID, COLUMN_NUM_MOVIMIENTOS, COLUMN_MOVIMIENTOS, COLUMN_OBSERVACIONES:** Todos estos atributos son de tipo *String*. Indican el nombre de las distintas columnas de la tabla utilizada, en este caso, las de la tabla *trayectos*.

Estos atributos se definieron, ya que, si se realiza alguna modificación respecto al nombre de la BBDD o de los elementos que la forman, únicamente se modificarán estos atributos, y no será necesario modificar las funciones que componen a esta clase.

Esta clase dispone de las siguientes funciones:

Constructor:

Dispone de un constructor para instanciar el objeto, al cual se le pasa como atributo el contexto de la actividad que lo instancia. Este constructor, llamará a su vez al constructor de la clase de la que se hereda, que generará la BBDD, si esta no existe, y establecerá una conexión con BBDD, en el objeto instanciado.

onCreate:

Esta función es heredada, pero es sobrescrita. Es llamada si la base de datos acaba de ser creada. En esta función se realiza la sentencia de *SQLite* que, mediante el uso de los atributos de esta clase, genera la tabla *trayectos* en la BBDD *Trayectos*.

onUpgrade:

Esta función es heredad, pero es sobrescrita. Es llamada cuando se modifica la versión de la base de datos. En esta función, se realiza la sentencia que elimina la tabla *trayectos,* si esta existe. Por último, se llama a la función *onCreate* para generarla de nuevo, con las nuevas características.

newTray:

Recibe como argumento un objeto de la clase *tray*. Esta se encargará de recoger los atributos de dicho objeto y utilizarlos para realizar una sentencia de inserción sobre la BBDD en la tabla *trayectos*. Está devolverá true si todo se realizó de forma correcta, y false, si sucedió alguna excepción durante el almacenamiento.

listarTray:

No recibe ningún argumento. Esta creará un *vector* de objetos de tipo *tray*, el cual, llenará con todos los trayectos de la BBDD, que obtiene realizando una consulta sobre esta. Al finalizar la función, esta devolverá dicho *vector*, independientemente de si el vector está vacío o no.

obtenerTray:

Recibe como argumento una variable de tipo *int* que indicará el ID de un trayecto. Esta función, instanciará un objeto de la clase *tray*, realizará una consulta a la BBDD para obtener el trayecto con el ID indicado, y almacenará el resultado de dicha consulta en el objeto de tipo *tray* creado. Al finalizar la función, devolverá el objeto de tipo *tray,* con los datos del trayecto cuyo ID coincida con el indicado en los argumentos.

modificarTray:

Recibe como argumento un objeto de tipo *tray.* Esta se encarga de coger los datos de dicho objeto, y realizar la sentencia de actualización del trayecto cuyo ID coincida con el del objeto de tipo *tray* recibido por parámetro. Si existe el trayecto con dicho ID en la tabla *trayectos*, se realizará la actualización de sus datos y se devolverá true. En caso contrario, no se llevará a cabo la actualización, ya que el trayecto no existe, y se devolverá false.

borrarTray:

Recibe como argumento una variable de tipo *int* que indicará el ID de un trayecto. Se encargará de realizar sentencia de borrado contra la BBDD, que eliminará aquella tupla de la taba *trayectos* cuyo ID coincida con el especificado. Si el borrado se realiza de forma correcta, se devolverá true, en caso contrario false.

4.4.1.2. Gestor de comunicaciones Aplicación móvil

Este subcomponente, es el encargado de llevar acabo de las comunicaciones con el *Servidor.* Ofrece una interfaz que permite, al resto de componentes de la *Aplicación móvil*, enviar peticiones y recibir respuestas del *Servidor*, únicamente sabiendo que para ello se utiliza una dirección IP. Este componente se encuentra almacenado en el paquete denominado *Connection*, en el cual se encuentra la única clase de este componte, *connection_manager.*

4.4.1.2.1. Connection_manager

Esta clase se encarga de realizar todas las operaciones, mediante el uso de *sockets* de *java,* para poder llevar acabo las comunicaciones con el *Servidor*. Además, ofrecerá una serie de funciones al resto de componentes de la *Aplicación móvil*, para que estos solo tengan que indicar la dirección IP del *Servidor* al que se quieren conectar. Una vez establecida la conexión, solo deberán indicar que petición quieren enviar y que petición esperan recibir, sin necesidad de interactuar directamente con los *sockets* de la conexión.

Esta clase dispone de los siguientes atributos:

- **Sock_server**: Objeto de tipo *socket*. Se utilizará para establecer la conexión con el Servidor.
- **stream_out:** Objeto de tipo *OutputStream*. Se generará a partir de *sock_server*, y se utilizará para la creación de canales de envío de datos al *Servidor*.
- **slient_mensaje:** Objeto de tipo *DataOutputStream*. Se genera a partir de *stream_out*, representa el canal de envió de datos utilizado para comunicarse con el *Servidor* y se utilizará para enviar mensajes a este.
- **in_Stream**: Objeto de tipo *DataInputStream*. Representa el canal de entrada de datos procedentes del *Servidor* y se utilizará para recoger las respuestas recibidas del *Servidor*.
- **ip_server**: Variable de tipo *String.* Almacenará la dirección IP del servidor a conectarse. Tomará como valor inicial "".
- **status_mod**: Variable de tipo *boolean*. Se utilizará para indicará si durante la conexión se ha seleccionado un modo de control. Tendrá el valor inicial false.
- **status_camera**: Variable de tipo *boolean.* Se utilizará para indicar si durante conexión se ha iniciado la cámara. Tomará como valor inicial false.
- **conection**: Variable de tipo *boolean.* Se utilizará para indicar si ya se ha establecido conexión con el *Servidor* o si la conexión sigue activa. Tomará como valor inicial false.

Todos estos atributos son privados.

Esta clase dispone de las siguientes funciones:

Constructor

Permite la instanciación del objeto. Recibe como argumento un variable de tipo *String*, esta indicará la dirección IP del *Servidor* al que se va a realizar la conexión. Este argumento se almacena en el atributo *Ip_server*.

Getters y setter

Permiten acceder y modificar atributos privados. En esta clase, se han generado para los atributos *status_mod* y *status_camera*.

conetionToServer

Función que recibe como argumento de entrada, dos variables de tipo *String,* estas indican: el modo de control seleccionado, en lenguaje natural y en el formato de petición entendido por el *Servidor*.

Esta función se encargará de inicializar *socket* de la comunicación, *Sock_server*, con la dirección IP almacenada en *Ip_server* y el puerto 8002. Una vez inicializado el socket, se extrae de él *in_stream* y *Stream_out,* del cual se extrae *Client_mensaje.*

Una vez se tienen todos los elementos de la conexión, se indica que ya se ha establecido la conexión, modificando el atributo *conection* a true. Después, se realiza la petición de selección de modo de control mediante *sendMessageToServer*, introduciendo como argumentos de entrada: la petición de selección de modo recibida como argumento de entrada al iniciar esta función, tanto en lenguaje natural como en formato de petición, y como respuesta esperada "MOD: OK\n". Si esta petición se realiza de forma correcta, se modifica el atributo *status_mod* a true.

Por último, independientemente de si se realiza de forma correcta la petición de selección de modo, si no sucede ninguna excepción, se devuelve true. Si no se puede establecer la conexión o se produce una excepción, se indicará mediante una impresión por pantalla en la terminal de Android y se devolverá false.

sendMessageToServer

Recibe como argumentos de entrada, tres variables de tipo *String*, que indican: Petición que se quiere realizar con el formato correspondiente, respuesta esperada y la petición a realizar en lenguaje natural.

Esta función, antes de nada, comprobará que exista una conexión, comprobando el valor de *conection*. Si existe conexión, se enviará a través de *Client_mensaje* la petición al *Servidor* y esperará una respuesta de 9 bytes en *In_Stream,* utilizando la función *readfully* de *In_Stream*.

Esta función devolverá un valor entero entre 1 a 3:

- Devolverá 1, si todo se realiza de forma correcta y la respuesta recibida en In_Stream corresponde con la esperada.
- Devolverá 2, si sucede alguna excepción o no hay establecida una conexión, indicándolo previamente con una impresión por terminal.
- Devolverá 3, en caso de que la respuesta recibida en *In_Stream* no fuera la esperada, indicándolo también por terminal.

receiveToServer

Esta función se utiliza para recibir una petición del *Servidor* y devolverla en un *String***.** No recibe ningún argumento de entrada. Esta función, esperará en *inStream* la recepción de una respuesta de 9 bytes del *Servidor,* utilizando la función *readFully.* Una vez tenga la petición, la transformará a *String* y la devolverá. Si sucede una excepción, devuelve un *String* vacío ("").

connection_close

Esta función no recibe argumentos. Esta función, antes de nada, comprobará que exista una conexión, comprobando el valor de *conection*.

Si existe conexión, se comprobará si la cámara se ha activado mediante *status_camera* o si se ha seleccionado un modo de control mediante *status_mod*. Si se da alguno de los dos, se utiliza *sendMessageToServer* para enviar al *Servidor* la petición correspondiente para desactivar la cámara o salir del modo de control según corresponda, y se modificará el valor de los atributos antes mencionados a false. Por último, se realizará el cierre de la conexión y la modificación del atributo *conection* a false.

Esta función devolverá un valor booleano, devolviendo el valor false, cuando al realizar la comprobación de conexión esta no exista o no se pueda cerrar la conexión, y se devolverá el valor true, cuando todo suceda de forma correcta.

4.4.1.3. Actividades aplicación

Este subcomponente, está compuesto por todas las clases de Java que se encargan de la lógica detrás de todas las pantallas de la *Aplicación móvil*. Estas utilizarán los otros dos subcomponentes, que se han definido anteriormente, para llevar acabo las funcionalidades que ofrece la *Aplicación móvil*. A continuación, se realizará una explicación de la implementación de estas clases.

4.4.1.3.1. SplashScreen.java

Es la lógica detrás de la pantalla *SplashScreen*, la cual se encargará de pasar a la pantalla *Principal* pasados 3sg.

Esta clase tiene los siguientes atributos:

- **SPLASH_SCREEN_DELAY:** Es una constante de tipo entero con valor 3000. Indicará en milisegundos el tiempo establecido para pasar a la pantalla *Principal*.

Esta clase dipone de las siguientes funciones:

onCreate

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. En esta función se añade el código para crear e instanciar una *TimerTask* que se lanzará tras *SPLASH_SCREEN_DELAY* milisegundos. Esta *TimerTask* se encargará de lanzar la pantalla *Principal* y finalizar esta, para no poder volver a ella.

4.4.1.3.2. Principal.java

Se encarga de la lógica de detrás de la pantalla *Principal*, la cual se encargará de pasar a la pantalla *ManualControl*, cuando se pulse el botón *Control manual,* y a la pantalla *AutomaticControl*, cuando se pulse el botón *Control automático*.

Esta clase dispone de las siguientes funciones:

onCreate

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. En este caso, no se añadirá código a esta función.

buttonManualControl

Función que se ejecuta mediante un *onClick* en el botón *Control manual.* Esta función generará el *AlertDialog* con las características especificadas en 4.3.3 MANUALCONTROL.

Además, también se encargará de llevar acabo las funcionalidades de los botones de dicho *AlertDialog*. Cerrando el cuadro de dialogo, al pulsar el botón *cancelar*, y pasar a la pantalla *ManualControl* con la dirección IP especificada en el *AlertDialog*, al pulsar el botón *aceptar*. Para pasar la dirección IP, se añadirá al *Intent* encargado de lanzar la pantalla *ManualControl.*

buttonAutomaticControl

Función que se ejecuta mediante un *onClick* en el botón *Control automático*. Esta será la encargada de pasar a la pantalla *AutomaticControl*.

4.4.1.3.3. ManualControl.java

Se encarga de la lógica de detrás de la pantalla *ManualControl*, la cual se encargará de establecer la conexión, enviar las peticiones y recibir las respuestas del *Servidor.* Además, se encargará de mostrar las imágenes capturadas por la cámara conectada al vehículo, las cuales, también transmite el *Servidor*.

Esta clase dispone de los siguientes atributos:

- **connection:** Es un atributo de tipo *Connection_manager*. Se utiliza para gestionar la conexión y envió de mensajes con el *Servidor*.

Esta clase dispone de las siguientes funciones:

onCreate

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. A esta función se le añade código, el cual realizará lo siguiente:

- Modificará las políticas de la pantalla para permitir la creación de *sockets*.
- Inicializa el atributo *connection,* con la dirección IP que se indicó en el *Intent* que lanzo esta pantalla.
- Se ejecutarán las funciones *conectionToServer*, *assingListenerCameraToogleButton*, *assingListenerDirecctionButtons* y *assingListenerCameraDirectionButtons*.

onbackPressed

Esta función es llamada cuando es pulsado el Button back físico del dispositivo móvil sobre el que se ejecuta la *Aplicación móvil*. A esta función se le añade el código que se encarga de finalizar toda comunicación con el *Servidor*, llamando a la función *connection_close* del atributo *connection*.

onDestroy

Esta función es llamada cuando la aplicación es expulsada de la memoria principal del dispositivo móvil sobre el que se ejecuta la *Aplicación móvil*. A esta función se le añade el código que se encarga de finalizar toda comunicación con el *Servidor*, llamando a la función *connection_close* del atributo *connection*.

connectionToServer

Esta función no recibe argumentos de entrada. Al iniciar esta función, se inicializa la conexión con el *Servidor*, llamando a la función *conectionToServer* del atributo *connection*, indicando como atributos de entrada, la selección de modo en formato de petición "MOD: MAN\n" y en lenguaje natural "Modo manual".

Por último, se comprueba el resultado de la función anterior. Si ha sucedido algún error en la comunicación o no se ha podido realizar la petición de selección de modo deseado, se genera una notificación de tipo *Toast* indicando "Error de conexión con el servidor" y se llama a la función *onbackPressed*. Si no ha sucedido ningún error, se finaliza la función.

assingListenerDirecctionButtons y assingListenerCameraDirectionButtons.

Ninguna de las dos funciones recibe argumentos de entrada. Estas dos funciones realizan las mismas operaciones sobre botones distintos. Se mantuvieron en dos funciones separadas para diferenciarlas de forma más sencilla.

Estas funciones se encargan de asignar *listeners* de tipo *OnTouch* a los botones encargados de la realización de peticiones de movimientos al vehículo y a la cámara. Este tipo de *listener*, permiten indicar que operaciones se quieren realizar cuando se pulsa y deja de pulsar un elemento de la interfaz a los que han sido asignados.

Por lo tanto, en los *listeners* asignados con estas funciones, se indicará que:

- Al ser pulsados, se llame a la función *sendMessageToServer* de esta clase, indicando como parámetros de entrada la petición a realizar con el formato correspondiente, la respuesta esperada y la petición realizada en lenguaje natural.
- Al dejar de ser pulsados, se indicará que se llame a la misma función *sendMessageToServer*, con los parámetros de entrada que se encargan de detener el movimiento realizado al pulsarse el botón.

assingListenerCameraToggleButton

Esta función no recibe parámetros de entrada. Es la encargada de asignar un *listener* de tipo *OnCheckedChange* al *ToggleButton* de activación/desactivación de la cámara, precisamente para que active/desactive la cámara al ser pulsado. Este tipo de *listener* permite indicar que acciones se quiere realizar cuando cambien estado un *ToggleButton* a *isChecked* o a *unChecked*.

Por lo tanto, el *listener* asignado mediante esta función, indicará que:

- Si el *ToggleButton* pasa a estado *isChecked*, se realizará una petición de activación de cámara mediante la función *sendMessageToServer* de esta clase. Una vez hecho esto, se comprueba el resultado de la función *sendMessageToServer*, y si el resultado confirma que se ha llevado a cabo de forma correcta, se establecerá como visible el *WebView* del *background* de esta pantalla, y, mediante la función *loadUrl* del *WebView,* se cargará en dicho *WebView* la dirección http://connection.getIp_server():8001.
- Si el *ToggleButton* pasa a estado *unChecked,* se enviará una petición de desactivación de cámara con la función *sendMessageToServer* de esta clase. Una vez hecho esto, se comprueba el resultado de esta función, y si el resultado confirma que se ha llevado a cabo de forma correcta, se ocultará el *WebView.*

sendMessageToServer

Esta función recibe como parámetro de entrada: Una petición que se quiere enviar al *Servidor* con el formato correspondiente, la respuesta esperada y la petición en lenguaje natural.

Esta función tomará estos mensajes y los enviará al *Servidor* mediante la función *sendMessageToServer*, pero la del atributo *connection*. Una vez hecho esto, se comprobará el resultado devuelto por dicha función:

- Si se recibe la respuesta esperada, se cerrará la función.
- Si no se recibe la respuesta esperada, se informará mediante una notificación de tipo *Toast* con el texto "Error al realizar (petición en lenguaje natural)".
- Si se ha producido un fallo de conexión, se informará con dos notificaciones de tipo *Toast* con el texto "Error al realizar (petición en lenguaje natural)" y "Error de conexión con el servidor". Por último, se ejecuta la función *onBackPressed*.

4.4.1.3.4. AutomaticControl.java

Se encarga de la lógica de detrás de la pantalla *AutomaticControl*, la cual se encargará de: Pasar a la pantalla *NewTray*, cuando se pulse el botón *Nuevo trayecto*, y a la pantalla *ExecuteTray*, cuando se pulse el botón *Ejecutar Trayecto*.

Esta clase no tiene atributos, pero dispone de las siguientes funciones:

onCreate:

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. En este caso, no se añadirá código a esta función.

buttonManualControl.

Función que se ejecuta mediante un *onClick* en el botón *Nuevo trayecto*. Esta será la encargada de pasar a la pantalla *NewTray*.

buttonAutomaticControl

Función que se ejecuta mediante un *onClick* en el botón *Ejecutar trayecto*. Esta será la encargada de pasar a la pantalla *ExecuteTray*.

4.4.1.3.5. NewTray.java

Se encarga de la lógica de detrás de la pantalla *NewTray,* la cual se encargará de: El almacenamiento de un nuevo trayecto en la BBDD, añadir movimientos a un trayecto mediante los botones del *especificador de movimientos*, generar el *listado de movimientos* de forma dinámica y realizar el borrado de un movimiento al pulsar el botón borrar movimiento del *listado de trayectos*.

Esta clase dispone de los siguientes atributos principales:

- **table_movements**: Objeto de tipo *TableLayout*. Representará el *listado de movimientos* en la interfaz gráfica. Al modificar este objeto, se modificará el *listado de movimientos.*
- **rows:** Vector de tipo *TableRow*. Almacenará la representación gráfica de cada movimiento especificado para el trayecto.
- **movements:** Vector de String. Almacenará cada movimiento especificado para el trayecto.
- **trayDB:** Objeto de tipo trayDB. Se utilizará para interactuar con la BBDD.

Las funciones de las que dispone esta clase son:

onCreate

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. A esta función se le añade código, el cual realizará lo siguiente:

- Enlaza el *listado de movimientos* a *table_movements*.
- Se inicializan los vectores *rows* y *movements.*
- Se elimina el contenido gráfico de *table_moments.*

onCreateOptionsMenu

Función ofrecida por Android que permite especificar un estilo especial de *ActionBar,* en este caso, se le añade el botón *confirmar*.

onOptionsItemSelected

Función ofrecida por Android, la cual, es llamada cuando se selecciona un botón del *ActionBar.* Anteriormente no se ha utilizado, debido a que solo se disponía del *Button back*, que por defecto llama a la función *onBackPressed*.

En esta función, se comprueba que el botón pulsado, es el botón *confirmar*.

Si es así, lo primero que se hace, es comprobar que el atributo *movements* tiene un tamaño mayor que 0 y menor que 1000. Dependiendo del resultado de la comprobación se realizará:

- Si no se cumple, para cada uno de estos casos, se muestra una notificación de tipo *Toast*. Está indicará, que el trayecto al menos debe tener un movimiento o que el máximo de movimientos es 999. Una vez generada la notificación, se finaliza la función, ya que el trayecto debe de cumplir esas dos condiciones para ser almacenado.
- Si se cumple la comprobación, se muestra un *AlertDialog* con las características especificadas en 4.3.5 NEWTRAY. Dependiendo del botón pulsado de dicho AlertDialog se realizará:
    - Si se selecciona el botón *aceptar*, se instancia un nuevo objeto de tipo *tray* con los datos del trayecto. Una vez creado objeto de tipo *tray*, se inserta el en la base de datos, realizando esto mediante la función *newTray* del atributo *trayDB*, indicando como argumento de entrada el nuevo objeto *tray*. Si la inserción se realiza con éxito, se genera una notificación de tipo *Toast* que lo informe. Por último, se llama a la función *onBackPressed*.
    - En caso de pulsar el botón *cancelar* del *AlertDialog*, este se cerrará.

Si el botón pulsado de la *ActionBar* no es el botón *confirmar*, se realiza la función *onBackPressed*.

go_left

Función que es ejecutada mediante un *onClick* al seleccionar el botón ← del *especificador de movimientos*. Esta función solo realiza una llamada a la función *generateNewRow*, pasándole como argumento de entrada 1.

go_right

Función que es ejecutada mediante un *onClick* al seleccionar el botón → del *especificador de movimientos*. Esta función solo realiza una llamada a la función *generateNewRow*, pasándole como argumento de entrada 2.

go_front

Función que es ejecutada mediante un *onClick* al seleccionar el botón ↑ del *especificador de movimientos*. Esta función solo realiza una llamada a la función *generateNewRow*, pasándole como argumento de entrada 3.

stop

Función que es ejecutada mediante un *onClick* al seleccionar el botón □ del *especificador de movimientos*. Esta función solo realiza una llamada a la función *generateNewRow*, pasándole como argumento de entrada 4.

generateNewRow

Función que recibe por atributos de entrada una variable de tipo entero, que tomará los valores del 1 al 4. Estos valores significarán lo siguiente:

- el valor 1, significará que el movimiento a añadir es *Girar a la izquierda*.
- el valor 2, significará que el movimiento a añadir es *Girar a la derecha.*

- el valor 3, significará que el movimiento a añadir es *Seguir de frente.*
- el valor 4, significará que el movimiento a añadir es *Parar*.

Lo primero que realiza esta función, es generar un objeto de tipo *TableRow* al que se le añade la imagen y texto correspondiente del movimiento que se va añadir. Además, también se añade a *movements* este movimiento, que tendrá el formato correspondiente para que lo comprenda el *Servidor*.

Una vez hecho esto, se le añade al *TableRow* el botón borrar de dicho movimiento, al cual, se le asigna un *listener* de tipo *onClick.* Este *listener*, hará que, al ser pulsado dicho botón, se elimine el movimiento de *movements*, *table_movements* y *rows,* y forzará la recarga de la interfaz del *listado de trayectos*, para que se elimine el trayecto del listado. Se realiza la asignación del *listener* antes de insertarlo en *table_movements* y *rows,* debido a que el usuario no lo visualizará, y, por lo tanto, no podrá interactuar con él hasta que se inserte en *table_movements.*

Por último, se añade el *TableRow* generado, a rows y *table_movements*, añadiéndose así a la interfaz del *listado de movimientos*.

4.4.1.3.6. ExecuteTray.java

Se encarga de la lógica de detrás de la pantalla *ExecuteTray,* la cual se encargará de: Listar todos los trayectos de la base de datos, generar el listado de forma dinámica, dar paso a las pantallas *EditTray* y *ExecutingTray*, con los datos de un trayecto, y eliminar un trayecto de la BBDD.

Esta clase dispone de los siguientes atributos principales:

- **table_trays:** Es un objeto de tipo *TableLayout*. Representará la interfaz gráfica de la pantalla cuando haya trayectos en la BBDD. Al modificar este objeto, se modificará el listado de trayectos.
- **rows:** Vector de objetos de tipo *TableRow*. Almacenará la representación gráfica de cada trayecto del listado de trayectos.
- **trays:** Vector de objetos de tipo *tray*. Almacenará cada trayecto en un objeto de tipo *tray.*
- **trayDB:** Objeto de tipo *trayDB*. Se utilizará para interactuar con la BBDD.

Las funciones de las que dispone esta clase son las siguientes:

onCreate:

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. A esta función se le añade código que realiza las siguientes acciones:

- Enlaza *table_tray* con la interfaz de la pantalla.
- Se inicializa el atributo *rows*, mediante la función *listarTray* de *TrayDB*, la cual, obtiene todos los trayectos de la BBDD.
- Se elimina todo el contenido gráfico de *table_Tray.*
- Se llama a la función *generateTableTrays*.

generateTableTrays

Esta función no recibe parámetros de entrada. Se encargará de crear el listado de trayectos de esta pantalla, mediante la información almacenada en *trays*.

Para ello, lo primero que se comprueba es si *trays* no está vacío.

Si está vacío, se muestra mediante un *TextView*, en mitad de la pantalla, un texto que indica que aún no se han introducido trayectos y como introducir uno.

Si el listado, no está vacío, se va generando la tabla de trayectos. Para realizar esto, se utiliza para cada fila del listado un objeto de tipo *TableRow,* donde se van insertando los distintos elementos de la fila.

Se comienza por la cabecera, la cual, su *TableRow* tiene los campos de texto que se muestran en la cabecera del listado. Una vez rellenando el *TableRow* de la cabecera, se inserta en *rows* y *table_trays*, actualizando así la interfaz.

Para el resto de elementos del listado, se recorre el vector *trays,* y por cada trayecto se rellena su *TableRow* de la siguiente manera:

- Se introduce en tres campos independientes el ID, número de movimientos y observaciones de dicho trayecto.
- Se introduce el botón *Realizar trayecto* de dicho trayecto. A este botón se le asigna un listener de tipo *onClick.* Este *listener* hará que, al ser pulsado este botón, se produzca un *AlertDialog* con las características especificadas en 4.3.6 EXECUTETRAY para este botón. Los botones de esta *AlertDialog* tendrán el siguiente comportamiento:
    - Cuando se pulse el botón *aceptar,* se pasará a la pantalla *ExecutingTray*, cuyo *intent* que lo lanza, tendrá añadida la IP del *Servidor* especificada en *AlertDialog*, y el ID del trayecto seleccionado.
    - Cuando se pulse el botón cancelar, se cerrar el *AlertDialog* y se finalizará la función para dicho botón.
- Se introduce el botón *Editar Trayecto* de dicho trayecto. A este botón se le asigna un listener de tipo *onClick*. Este *listener* hará que, al ser pulsado este botón, se pase a la pantalla *EditTray*, cuyo *intent* que lo lanza, tendrá añadido el ID del trayecto seleccionado.
- Se introduce el botón *Borrar Trayecto* de dicho trayecto. A este botón se le asigna un listener de tipo *onClick.* Este *listener* hará que, al ser pulsado este botón, se produzca un AlertDialog con las características especificadas en 4.3.6 EXECUTETRAY para este botón. Los botones de esta *AlertDialog* tendrán el siguiente comportamiento:
    - Cuando se pulse el botón *aceptar,* se realizan las siguientes acciones en el orden en el que se citan:
        - Se borra el trayecto seleccionado de la BBDD mediante el uso de la función *borrarTray* de *trayDB*, pasándole como

parámetro de entrada el ID del trayecto seleccionado. Se elimina el
        - Se elimina el trayecto del vector *trays.*
        - Se comprueba si quedan trayectos en trays:
            - ❖ Si quedan trayectos, se elimina este trayecto de *table_trays,* y se oculta y muestra *table_trays*, forzando a la recarga del listado de trayectos.
            - ❖ Si no queda ninguno, se borra todo el contenido gráfico de *table_trays*, se fuerza a la recarga de la interfaz del listado de trayectos y se muestra el *TextView* que indica que la tabla está vacía.
        - Se elimina el trayecto del vector *rows*, y se genera una notificación de tipo *Toast* indicando el correcto borrado.
    - Cuando se pulse el botón cancelar, se cerrar el *AlertDialog* y se finalizará la función para dicho botón.
- Una vez introducido todos los elementos del *TableRow,* se inserta en *rows* y *table_trays*, actualizando así la interfaz.

Cuando se hayan insertado todos los trayectos en el listado, se cierra la función.

4.4.1.3.7. EditTray.java

Se encarga de la lógica de detrás de la pantalla *EditTray,* la cual se encargará de: Cargar los datos del trayecto a editar, la modificación de un trayecto en la BBDD, añadir movimientos a un trayecto mediante los botones del *especificador de movimientos*, generar el *listado de movimientos* de forma dinámica y realizar el borrado de un movimiento al pulsar el botón borrar movimiento del *listado de trayectos*.

Como se puede observar, esta clase realiza casi la misma lógica que *NewTray.java*. Además, comparten la mayoría de su implementación con pequeños cambios en las funciones y atributos de los que dispone. A continuación, se indicarán las diferencias entre estas dos clases.

Esta clase dispone de un atributo adicional:

- **Actual_tray:** Es un objeto de tipo *tray*. Almacenará los datos del trayecto que se va a modificar.

Las funciones que se han modificado o añadido en esta clase son las siguientes:

onBackPressed

No se utiliza la función por defecto, sino que se sobrescribe. En este caso, el código se encargará de lanzar la pantalla ExecuteTray de nuevo y se finalizar esta.

Esto se realiza así, ya que se quiere mostrar la información actualizada en *ExecuteTray*. Obligando de esta manera, a volver a realizar la consulta a la BBDD en dicha pantalla.

Por último, se finaliza esta pantalla, para que al volver a la pantalla *ExecuteTray* y se pulse el *Button back,* no se vuelva a esta pantalla.

onOptionsItemSelected

Se modifica ligeramente el comportamiento del botón *aceptar* del *AlertDialog* generado en esta función.

En este caso, no se generará un nuevo objeto *tray*, sino que se utilizará el atributo *actual_tray* al que se le actualizarán los datos. Por otro lado, no se utilizará la función *newTray* de *TrayDB*, sino la función *modificarTray* de *TrayDB*, pasándole *actual_tray* como argumento de entrada. Por último, se cambia la notificación *Toast* de la generación correcta del trayecto, por una que indica la modificación correcta del trayecto.

onCreate

En este caso, lo primero que se realiza es la obtención de los datos del trayecto a modificar y su almacenamiento en *actual_tray*. Para llevar acabo esto, se utiliza la función *obtenerTray* de *TrayDB*, indicando como argumento de entrada el ID especificado en el *Intent* que lanzo esta pantalla.

Una vez hecho esto, se comprueba que *actual_tray* no este vacío. Si no está vacío, se llevan a cabo las mismas acciones que en *NewTray.java*. Si está vacío, se llama a la función *onBackPressed*.

generateTableMovemts

Esta función solo está disponible en esta clase. No recibe ningún argumento de entrada. Esta función utiliza los datos de *actual_tray* para generar el *listado de trayectos*. Para ello, obtiene los movimientos almacenados en *actual_tray,* comprueba uno por uno los movimientos, y por cada movimiento ejecuta la función *generateNewRow*, pasándole como argumento de entrada el número correspondiente a dicho movimiento.

4.4.1.3.8. ExecutingTray.java

Se encarga de la lógica de detrás de la pantalla *ExecutingTray,* la cual se encargará de: Listar los movimientos del trayecto ejecutado, realizar las peticiones para indicar al *Servidor* los movimientos del trayecto, actualizar el estado de los movimientos y mostrar las imágenes capturadas por la cámara del vehículo.

Dispone de los siguientes atributos:

- **connection**: Objeto de tipo *connection_manager*. Se utilizará para manejar la conexión con el *Servidor*.
- **actual_tray**. Objeto de tipo *tray*. Almacenará los datos del trayecto a realizar.
- **trayDB**: Objeto de tipo *trayDB*. Se utiliza para interactuar con la BBDD.
- **table_movements:** Es un objeto de tipo *TableLayout*. Representará la interfaz gráfica del *listado de movimientos*, donde se indicará el estado de los movimientos del trayecto a realizar.

- **rows:** *Vector* de objetos de tipo *TableRow*. Almacenará la representación gráfica de cada movimiento del *listado de movimientos*.
- **image_state_moves:** *Vector* de objetos de tipo *ImageView*. Almacenará las imágenes de estado de las filas del *listado de movimientos.*
- **move_state_receiver:** Objeto de tipo *thread.* Se utilizará para la recepción del estado de los movimientos. Es necesario para que no se bloquee la interfaz de la pantalla mientras se realiza la espera de los estados de los movimientos.

Las funciones de las que dispone esta clase son:

onCreate

Función de la que disponen todas las pantallas, la cual se ejecuta cuando la pantalla es lanzada. A esta función, se le añade el código para realizar las siguientes acciones en el orden que se citan:

- Modificación de las políticas para poder crear *sockets* en esta clase.
- Enlaza *table_movements* con el *listado de movimientos*.
- Se inicializan los vectores *rows* y *image_state_moves*.
- Se almacena en *actual_tray* el trayecto a realizar, para ello se usa la función *obtenerTray* de *trayDB ,* pasándole como atributo de entrada el ID especificado en el *Intent* que lanzo esta pantalla.
- Se comprueba el contenido de *actual_tray*. Dependiendo de este se realizará:
    - Si no tiene contenido, se llama a la función *OnbackPressed*.
    - Si tiene contenido, se realizan las siguientes acciones:
        - Se inicializa el atributo *connection* pasándole como parámetro al constructor de este, la dirección IP especiada en el Intent que lanzó esta pantalla.
        - Se llama a las funciones *conectionToServerAndCam* y *sebdAndGenerateMoves*.
        - Se inicializa el hilo *move_state_receiver,* indicándole que la función que tiene que realizar es *changeStateReceiver,* y se lanza dicho hilo.

onBackPressed

Se sobrescribe la función por defecto, añadiéndole el código que cierra la conexión con el *Servidor* mediante la llamada a la función *connection_close* del atributo *connection*.

onDestroy

Al igual que en *onBackPressed*, se sobrescribe la función por defecto, añadiéndole el código que cierra la conexión con el *Servidor* mediante la llamada a la función *connection_close* del atributo *connection*.

sendMessageToServer

Función que realiza exactamente las mismas acciones que la función con el mismo nombre en *ManualControl.java*.

conectionToServerAndCam

Misma función que *connectionToServer* de *ManualControl.java,* a la cual se le modifica la petición de selección de modo, indicando en este caso, la de modo automático "MOD: AUT\n" y en lenguaje natural "modo automático".

Además, se le añade una petición de activación de cámara mediante *sendMessageToServer*, comprobando el resultado de esta.

- Si la petición se realiza de forma correcta, se carga en el *WebView* del *visor de imágenes*, mediante la función *loadUrl* de dicho *WebView*, la dirección http://connection.getIp_server():8001.
- Si la petición no se puede realizar, se genera una notificación de tipo *Toast* indicando el fallo de conexión con el Servidor y se llama a la función *onBackPressed*, ya que es estrictamente necesaria la activación de la cámara.

generateNewRow

Prácticamente la misma función que *generateNewRow* de *NewTray.java*, la cual, en vez de añadir un botón de borrado de movimiento al *TableRow* generado por fila, añade un *ImageView*. Este *ImageView*, se encargará de mostrar la imagen de estado. Por último, se incluye el código para insertar el *ImageView* en *image_state_moves*.

sendAndGenerateMoves

Esta función no recibe parámetros de entrada. Al iniciarse esta función, enviará al *Sevidor,* una petición de indicación de movimientos y otra de número de movimientos, con el número de movimientos de *actual_tray*. Estas dos peticiones tendrán el formato adecuado, y se enviarán, mediante la función *sendMessageToServer* de esta clase.

Una vez enviadas las dos peticiones anteriores, se recorre los movimientos de *actual_tray,* y por cada movimiento se realiza:

- El envío de una petición al *Servidor* indicando el movimiento correspondiente, con el formato adecuado.
- Una llamada a la función *generateNewRow*, a la que se le pasará como argumento de entrada, el número de movimiento correspondiente al movimiento que se está tratando.

Si en esta función, al realizar alguna petición al Servidor, esta no se puede llevar acabo o no se recibe la respuesta esperada, se notificará mediante una notificación de tipo *Toast* el error sucedido, y se llamará a la función *onBackPressed*.

changeStateAndReceive

Función ejecutada por el hilo *move_state_receiver*. Esta función, no recibe parámetros de entrada. Esta función esperará por cada uno de los movimientos de *actual_tray*, la respuesta del *Servidor* del estado del movimiento, utilizando la función *receiveToServer* del atributo *connection*.

Cuando se reciba una respuesta del *Servidor* respecto al estado de algún movimiento y esta no esté vacía, se realizará las siguientes acciones:

- Se modificará en *image_state_moves*, la imagen de la fila del *listado de movimientos* dedica al estado del movimiento del que se recibió la respuesta. Esta imagen se cambiará a la del estado recibido.
- Si la respuesta fue un mensaje de error, se muestra una notificación de tipo *Toast* indicando el suceso y se ejecuta la función *onBackPressed*.
- Se fuerza la recarga de la interfaz del *listado de movimientos*, ocultándolo y volviendo a mostrar mediante el atributo *table_momvements.*

Si la respuesta recibida de *receiveToServer*, está vacía, significará que ha habido un error en la conexión. Por lo tanto, se muestra una notificación de tipo *Toast* indicando el suceso y se llama a la función *onBackPressed*.

En esta función, todas aquellas acciones que se realizan sobre la interfaz gráfica se ejecutan dentro de la función *runOnUiThread* de Android, ya que un hilo que no sea de interfaz no puede realizar cambios sobre esta. La función *runOnUiThread*, permite llamar al hilo encargado de la interfaz gráfica de la pantalla, e indicarle un fragmento de código para que lo realice.

4.4.2. Servidor

En este apartado, se hablará de la implementación realizada para el componente *Servidor*. Este está compuesto por varios subcomponentes, los cuales son: *Controlador de comunicaciones con Aplicación móvil*, *Controlador de comunicaciones con Controlador*, *Servidor*, *Procesador de imágenes*, *Capturador de imágenes* y *Clasificador de imágenes*. Salvo el capturador de imágenes, el resto de componentes están programado en C++.

Este componente, llevará acabo las siguientes funcionalidades principales:

- Capa intermedia entre *Controlador* y *Aplicación móvil*.
- Toma de decisiones en el modo de *Control automático*, para realizar la conducción autónoma del vehículo.
- Transmisión de las imágenes capturadas mediante la cámara, en los modos de *Control manual* y *Control automático*.

A continuación, se explicará la implementación de cada uno de sus subcomponentes.

4.4.2.1. Controlador de comunicaciones con la Aplicación móvil

Es el subcomponente que se encarga de manejar la comunicación con la *Aplicación móvil* mediante el uso de sockets. Está compuesto por el archivo *ConAplicacion.cpp*.

El código de este subcomponente tiene únicamente implementada una clase de C++ denominada *conAplicacion*. Esta ofrecerá una serie de funciones para manejar de forma simple una comunicación Servidor-Cliente, sin necesidad de que el subcomponente que lo utilice tenga que tratar con los *sockets* de la comunicación.

Esta clase dispone de los siguientes atributos.

- **sock_fd_server:** Variable de tipo *int.* Almacena el descriptor de fichero del *socket* del servidor de la comunicación. Toma el valor inicial -1.
- **sock_fd_client:** Variable de *int*. Almacena el descriptor de fichero del *socket* del cliente de la comunicación. Toma el valor inicial -1.
- **Server_sock_addr y client_sock_addr:** Estructuras de tipo *sockaddr_in*. Almacenarán las direcciones de los *sockets* de la comunicación. En estos objetos se podrá encontrar la dirección IP y el puerto de los *sockets.*

Las funciones que ofrece esta clase son:

Constructor y destructor por defecto

Ofrece el constructor y destructor por defecto, los cuales, no realizan ninguna acción adicional, salvo la instanciación y eliminación de los datos de la clase, respectivamente.

Contructor con argumentos de entrada.

Este constructor, recibe como atributo de entrada una dirección IP, esta se pasará en un array de *char* de 16 posiciones, con formato *a.b.c.d*.

En este constructor, se realizarán las siguientes acciones para establecer el socket que corresponde con el servidor de la comunicación, y escuchar nuevas conexiones:

- Se obtiene un descriptor de fichero para el *socket* del servidor de la comunicación y se almacena en *sock_fd_server*. Para ello, se indican a la función *socket*, los siguientes parámetros de entrada:
  - El dominio de la dirección del *socket*, en este caso, *AF_INET* [B49]. Esto se debe, a que el socket se utilizará para comunicar dos procesos que se ejecutan en distintas máquinas.
  - El tipo de protocolo utilizado por el *socket*, en este caso, *SOCK_STREAM*. Indicará que el protocolo usado es *TCP*.
- Se convierte la dirección IP recibida por parámetros, en formato a.b.c.d, a formato binario, almacenándose en una variable auxiliar de tipo *in_addr*.
- Se rellena el atributo de la clase *Server_sock_addr*, indicando el dominio de la dirección *AF_INET*, la dirección IP en formato binario y el puerto 8002 en formato *Big Endian*.
- Se realiza la función *bind*, pasándole como parámetro *Server_sock_addr.*
- Se realiza la función *listen*, para poder escuchar nuevas posibles conexiones a el *sokect* del servidor de la comunicación. A esta función, se le pasará *sock_fd_server* y *SOMAXCONN* como argumento de entrada. El segundo, es una constante, que indica el máximo de conexiones a un socket.

Si en alguna de las acciones se produce un error, se asigna a *sock_fd_server*, el valor -1 y se terminará el constructor. Indicando así, que no se ha podido establecer el *socket*.

getSock_server y getSock_client

Funciones para devolver los descriptores del cliente y el servidor de la comunicación. La primera devuelve el atributo *sock_fd_server*, y la segunda, el atributo *sock_fd_client*.

closeClient

Función que cierra la comunicación con el cliente. Para ello, comprueba que *sock_fd_client*, no es -1. Si la condición no se cumple, se finaliza la función. Si la condición se cumple, se llama a la función *close* pasándole *sock_fd_client*, como atributo de entrada, lo que cerrará el descriptor y la comunicación asociada a este.

closeServer

Función que cierra el canal para establecer conexiones con el *Servidor*. Realiza las mismas operaciones que *closeClient* pero con *sock_fd_server*, en vez de, *sock_fd_client*.

listenClient

Función que mantiene a la espera aquel que la ejecuta, hasta que se recibe una conexión al socket del servidor de la comunicación.

Para llevar acabo esto, se llama a la función *accept,* pasándole como parámetros de entrada *sock_fd_server* y *client_sock_addr.* Se esperará en esta función hasta que se establezca una conexión a *sock_fd_server*. Una vez se establezca, esta función almacenará en *client_sock_addr* los datos de la dirección del cliente y devolverá el descriptor asociado a esta nueva conexión, que se almacena en *sock_fd_client*. Como resultado de esta función, se devolverá *sock_fd_client*.

enviar

Función que se utilizan para enviar una respuesta al cliente. Recibe como parámetros de entrada un buffer de tipo *char* y una variable *int* que indicará el número de *bytes* a enviar de dicho buffer.

Esta función, utilizará la función *write,* que se utiliza para escribir *bytes* en un socket*,* pasándole como atributo de entrada *sock_fd_client*, el *buffer* y la variable que indica el número de *bytes* a enviar.

Debido a que la función *write* devuelve el número de bytes enviados, se comprueba si se han enviado el número de bytes deseados. Se realizará la operación *write* tantas veces como sea necesario, hasta que la suma de los números de bytes devueltos por dicha función corresponda con los que se deseaban enviar.

recibir

Función que se utiliza para esperar y recoger una petición del cliente de la comunicación. Recibe como parámetros de entrada un buffer de tipo *char* y una variable *int* que indicará el número de *bytes* que se esperan recibir y almacenar en el *buffer*.

Realiza las mismas operaciones que Enviar, pero utilizando la función *read*, que se utiliza para leer *bytes* de un socket. En este caso, se comprobarán que *read* devuelve el número de bytes que se deseaban leer.

4.4.2.2. Controlador de comunicaciones con Controlador

Es el subcomponente que se encarga de manejar la comunicación con el *Controlador* mediante el uso del puerto serial. Está compuesto por el archivo *ConControlador.cpp*.

Este subcomponente, utiliza una librería desarrollada por terceros para abrir y enviar/recoger datos a través del puerto serial. Esta librería es la siguiente:

wiringPi

[B50] Es una librería de código abierto, desarrollada específicamente para las placas Raspberry Pi, en el lenguaje de programación C. Esta ofrece funcionalidades desde el uso de los pines *GPIO* a los puertos seriales, entre otras. Se decidió utilizar esta librería por los siguientes motivos:

- Debido a que está escrita en C, puede ser utilizada en C++, que es el lenguaje utilizado para codificar el componente *Servidor*.
- Ofrece una librería especifica llamada *wiringSerial*, la cual permite abrir y cerrar conexiones a los puertos seriales. Además, dispone de funciones para enviar y recibir mensajes a través de dicho puerto.
- Si fuera necesario, permite el uso de forma sencilla de los puertos GPIO de la Raspberry utilizada para ejecutar el *Servidor*.

El código de este subcomponente tiene únicamente implementada una clase de C++ denominada *conControlador*. Esta ofrecerá una serie de funciones para manejar de forma simple una comunicación mediante el puerto serial, mediante el uso de las funcionalidades ofrecidas por *wiringSerial*.

La clase *conControlador*, dispone de los siguientes atributos:

- **fd_arduino:** Variable de tipo *int*. Almacenará el descriptor de fichero asignado a la conexión con el *Controlador*. Se inicializa con el valor -1.

Esta clase dispone de las siguientes funciones:

Constructor y destructor por defecto

Ofrece el constructor y destructor por defecto, los cuales, no realizan ninguna acción adicional, salvo la instanciación y eliminación de los datos de la clase, respectivamente.

Constructor con argumentos de entrada

Dispone de un constructor que recibe como argumento de entrada una variable de tipo *String*. Esta variable, indicará la dirección del puerto serial del *Controlador*.

Esta función, mediante el uso de la función *serialOpen* de la librería *WiringPi*, abrirá una conexión al puerto serial del Controlador y devolverá el descriptor de fichero de dicha conexión. Para llevar esto a cabo, se le pasa a la función *SerialOpen* la dirección recibida como argumento de entrada en esta función. El descriptor de la conexión establecida se almacenará en *fd_arduino*.

con_fd

Función que devuelve el descriptor de fichero de la conexión establecida con el *Controlador*. Para ello, devuelve el valor de *fd_arduino*.

closeArduino

Función que cierra la conexión establecida con el *Controlador*. Para ello llama a la función *SerialClose* de *WiringPi*, la cual, cierra conexiones a puertos seriales, pasándole como argumento de entrada *fd_arduino*.

enviar_arduino

Función que envía 9 bytes al *Controlador*, a través de un puerto serial. Envía justo este tamaño, debido a que todas las peticiones reconocidas por el *Controlador* tienen este tamaño. Esta función recibe como argumento de entrada un array de *char* de 9 posiciones

Para poder cumplir esta función, se utiliza la función *serialPuts* de la librería *WiringPi*. La cual, envía a través de un descriptor de conexión a un puerto serial, el contenido de un *buffer* de *char*. Se envían elementos de este buffer hasta que se encuentre el carácter '\0'.

Por lo tanto, para llevar acabo la función, se llama a la función *SerialPuts*, pasándole como argumentos de entrada, *fd_arduino* y el array de *char* recibido como argumento de entrada en esta función, al que se le habrá concatenado el carácter '\0'.

respuesta_arduino

Función que recibe 9 bytes del Controlador, a través de un puerto serial. Esta función recibe como argumento de entrada un array de *char* de 9 posiciones.

Para llevar a cabo esta función, se utiliza serialGetChar de la librería *WiringPi*. Al llamar a esta función, se le debe de pasar como argumento de entrada, el descriptor de fichero de una conexión a un puerto serial. Esta función esperará hasta que haya disponible un *byte* en la conexión especificada. Una vez que lo hay, lo recoge y lo devuelve.

Por lo tanto, para llevar a cabo esta función, se llama 9 veces a la función *serialGetChar,* indicándolo como argumento de entrada *fd_arduino.* Los bytes recogidos, se introducen en el array de *char* recibido como argumento de entrada de esta función.

Esta función devolverá el valor entero 0 si no ha habido ningún error. Si en alguna de las llamadas a *SerialGetChar* se obtiene el valor -1, indicará que ha sucedido un error, por lo que se informará mediante una impresión por pantalla, se introducirá en el array de *char* el valor "MSG: ERR\n" y se devolverá -1 en la función.

enviar_arduino_dir

Función auxiliar diseñada para controlar la dirección del vehículo en el modo de *control Automático.* Esta función permitirá enviar peticiones de giro del vehículo, indicando cuanto tiempo se quieren realizar.

Para llevar acabo esto, esta función recibe como argumentos de entrada: Un array de *char* de 9 posiciones, una variable entera que indicará la dirección de giro, y una variable entera que indica el tiempo de realización del movimiento en microsegundos.

Lo primero que realiza esta función, es una comprobación de la variable entera que indica la dirección de giro. Dependiendo de este valor, se realiza:

- Si el valor es 1, significará que se quiere girar a la izquierda. Por lo tanto, se almacenará en el array de *char* recibido como argumento, el valor "DIR: DER\n"
- Si el valor es 2, significará que se quiere girar a la derecha. Por lo tanto, se almacenará en el array de *char* recibido como argumento, el valor "DIR: IZQ\n"

Una vez hecho esto, se envía al *Controlador* el contenido del array de *char*, mediante la función *enviar_arduino*, pasándole como argumentos de entrada *fd_arduino* y el array de *char*.

Enviada la petición del movimiento que se desea realizar, se duerme el proceso tantos microsegundos como se haya especificado en el parámetro de entrada, mediante la función *usleep.*

Por último, se copia "DIR: STP\n" en el array de *char* recibido por parámetro y se envía mediante la función *enviar_arduino.*

Siempre que se realice un envió al *Controlador*, después se realiza la comprobación de la respuesta de este mediante *respuesta_arduino* y el array de *char*. Si *respuesta_arduino* devuelve -1 o el array de *char* tiene el valor "MSG: ERR\n", se devuelve el valor -1.

4.4.2.3. Capturador de imágenes

Es el subcomponente que se encarga de capturar las imágenes mediante el uso de la cámara conectada al vehículo. Además, también se encargará de almacenar dichas imágenes en el almacenamiento interno del dispositivo sobre el que se ejecuta. Por último, dará la posibilidad de transmitir las imágenes a una dirección de tipo *http*.

Este subcomponente ha sido totalmente desarrollado por terceros y se ejecuta como un proceso independiente.

El nombre real de este subcomponente es *Motion* [B51]. El cual está desarrollado en el lenguaje C y es de código abierto. Es una funcionalidad instalable en *SO's* Linux, la cual, permite capturar imágenes de forma continua, a través de una cámara conecta al dispositivo sobre el que se ejecuta el SO. Este permite una gran cantidad de configuraciones y dispone de la posibilidad de levantar un servidor *http*.

Se decidió utilizar esta funcionalidad, debido a los siguientes motivos:

- Esta funcionalidad trabaja bastante bien en placas con poca potencia como la Rasperry Pi. Ofreciendo un retraso entre la obtención de la imagen y el momento en el que ya está disponible para su tratamiento, de pocas décimas de segundo.
- Permite su lanzamiento en un proceso paralelo independiente al resto. Lo facilita la paralelización del componente *Servidor*.
- Permite almacenar de forma automática las imágenes capturadas mediante la cámara. A estas imágenes almacenadas, se les asigna un nombre que corresponde con la fecha del segundo en el que se capturaron. Lo que facilita en gran medida su reconocimiento. El formato asignado es:

  VV-YYYYMMDDHHXXSS-ZZ

  - VV: Indica el dispositivo mediante el que captura las imágenes, en este caso, al tener solo una cámara siempre será 01.
  - YYYY: Indicará el año en que se capturo la imagen.
  - MM: Indicará el mes en que se capturo la imagen.
  - DD: Indicará el día en que se capturo la imagen.
  - HH: Indicará la hora en que se capturo la imagen.
  - XX: Indicará el minuto en que se capturo la imagen.
  - SS: Indicará el segundo en que se capturo la imagen.
  - ZZ: Indicará dentro de dicho segundo, su número de creación. Siendo la imagen 00, la primera que se capturo.
- Permite lanzar un servidor *http* para transmitir las imágenes capturadas. Este servidor, se puede lanzar en la IP de la máquina y en el puerto que especifique el usuario.
- Permite establecer el tamaño y formato de las imágenes capturadas.
- Permite indicar en que directorio se quiere almacenar las imágenes.
- Permite indicar el *framerate* de captura de imágenes.

Por otro lado, se implementó un componente propio, que utilizando la librería *OpenCV,* realizaba la captura de imágenes. Pero en este componente, se producían retrasos muy grandes entre la realización de la captura, y el momento en el que ya se podía tratar, dicha captura. Estos retrasos, hacían imposible realizar una conducción autónoma medianamente fluida.

Por último, debido a que el fichero de configuración de este componente varía dependiendo de si se selecciona el modo de *Control automático* o *Control manual*. Se dispone de dos ficheros de configuración diferentes. A continuación, se indican los campos que son modificados en dicho fichero, para cada uno de los modos de control, con respecto al fichero de configuración por defecto de *Motion*.

Elementos comunes de *Control automático* y *Control manual*

- **daemon off:** Se indica, que la funcionalidad no se quiere inicializar al arrancar el sistema.
- **framerate 24**: Indica el número máximo de imágenes que se capturarán por segundo.
- **picture_type jpeg:** Indica que, el formato de las imágenes capturadas será *jpeg*.
- **emulate_motion on:** Indica que, aunque no haya variaciones entre la imagen actual y la última capturada, esta se almacene.
- **stream_port 8001**: Indica el puerto en el que se produce la transmisión de las imágenes.
- **stream_localhost off:** Indica que el servidor de transmisión de las imágenes se levanta en la dirección IP física del dispositivo sobre el que se ejecuta, en vez de *localhost* (127.0.0.0).

Modo de *Control automático*

- **width 320:** Indica el ancho de la imagen en pixeles.
- **Height 240:** Indica el alto de la imagen en pixeles.
- **output_pictures on:** Indicará que se almacenan las imágenes capturadas.

Modo de *Control manual*

- **Width 640:** Indica el ancho de la imagen en pixeles.
- **Height 360:** Indica el alto de la imagen en pixeles.
- **output_pictures off:** Indicará que no se almacenan las imágenes capturadas.

4.4.2.4. Procesador de imágenes

Este subcomponente se encargará de convertir las imágenes capturadas por *Capturador de imágenes* a un formato comprensible para *el Clasificador de imágenes*. Está compuesto por el archivo *ProcImage.cpp*.

Este subcomponente, utilizará una librería de terceros para abrir y poder procesar las imágenes capturadas por *Capturador de imágenes*. Esta librería, es la siguiente:

OpenCV

[B52] Es una librería de procesamiento de imágenes de código libre. Está desarrollada en C/C++ para ofrecer un buen rendimiento. Esta dispone de una gran cantidad de funciones implementadas para el tratamiento de imágenes. Se decide utilizar por los siguientes motivos:

- Dispone de objetos *Mat*, en los cuales, se puede realizar la carga de una imagen y tratarla de forma parecida a un array de 2 dimensiones.
- La librería está disponible para el lenguaje C++, en el que se encuentra implementado el *Servidor*.
- Dispone de funciones para la conversión de imágenes. Por ejemplo, convertir una imagen a escala de grises o a blanco y negro.

Este componente está formado únicamente por dos funciones. A continuación, se especifica su implementación.

obtain_image_name

Esta función se encarga de obtener el nombre de las imágenes capturadas por *Capturador de imágenes*. Para ello, devolverá el nombre que se le debería asignar a una imagen al capturarse en el segundo que se ejecuta esta función.

Para llevar acabo esto, obtiene mediante la función *localtime,* el tiempo del sistema justo cuando se ejecuta dicha función *localtime.* El tiempo obtenido se almacena en una estructura de tipo *tm*, utilizadas para este propósito.

Una vez hecho esto, se concatena en un *String* el contenido de *tm*. Para ello, lo primero que se hace es concatenar a este *String* el valor "01- ". Después, se van accediendo a los distintos atributos de esta estructura, y se van concatenado los valores en el *String* creado, asegurando que cumpla el formato de nombre asignado por el *Capturado de imágenes*, el cual, se especificó en 4.4.2.3 CAPTURADOR DE IMÁGENES.

Por último, se devuelve el *String* con toda la información del tiempo del sistema, con el formato del *capturador de Imágenes*.

min_image_BN

Esta función convertirá las imágenes recibidas, que tendrán un tamaño de 320x240, a una imagen de ese mismo tamaño, pero en blanco y negro. Después, esa imagen se reduce a una matriz de 40x30 y se introduce en un vector de 1200 posiciones, formato entendido por el *Calificador de imágenes*. En el apartado de dedicado al clasificador, se hablará en profundidad del motivo de esta conversión.

Para llevar a cabo esta función, se recibe como parámetro de entrada una variable *String* con la ruta y nombre de la imagen a tratar.

Esta función utilizará las siguientes variables para llevar a cabo el proceso de conversión:

- **image**: Objeto de tipo *Mat*. Almacenará el valor de la imagen original.
- **filter_image_BN**: Objeto de tipo *Mat*. Almacenará la imagen original convertida a blanco y negro, es decir, sus pixeles tendrán el valor 0 o el valor 255.
- **aux_matrix:** Matriz con 40 filas y 64 columnas de tipo *uchar*. Se utilizará para reducir la imagen. Se escoge de este tamaño, ya que 320/40 = 8 y 64/8 =8. Por lo que, con esta matriz se pueden hacer de 8 grupos de 64 bytes, que corresponde con 8 filas de la imagen.
- **image_code:** Vector de variables uchar. Almacenará la imagen reducida.

El funcionamiento de la función es el siguiente:

Lo primero que realiza, es la carga de la imagen a tratar, mediante la función *imread* de *OpenCV*, indicándole como argumentos de entrada, el *String* con la dirección y nombre de la imagen a tratar, y *CV_LOAD_IMAGE_GRAYSCALE*. Con estos atributos de entrada, la imagen se cargará en escala de grises en *image*.

Después, se comprueba si dicha imagen no está vacía, mediante la función *empty* de los objetos *Mat*.

Si no está vacía, se llevan a cabo las siguientes acciones:

- Realiza un filtrado de tipo *Otsu* [B53] de *image*, convirtiéndola en una imagen en blanco y negro, y almacenando el resultado en *filter_image_BN* Para ello, se llama a la función *threshold* de *OpenCV*, realizando la siguiente llamada:

```
threshold(image,filter_image_BN,0,255,THRESH_BINARY|CV_THRESH_OTSU);
```

Ilustración 64 - Llamada a thershold para realizar filtro Otsu

- Se inicializa el vector *image_code* con 1200 posiciones de tipo *uchar* y de declarar una variable auxiliar de tipo *int contador* para recorrerlo.
- Se recorre *filter_image_BN*, fila a fila realizando las siguientes acciones:
    - Por cada fila se ejecuta el siguiente código:

```
for(int ii = 0; ii < 320; ii++){
    /* Se recoje cada pixel */
    uchar pixel = filter_image_BN.at<uchar>(jj,ii);

    /* Se alamacena en su posición correspondiente de aux_matrix */
    aux_matrix [ii/8][(ii%8) + (8*aux_contador)] = pixel;
}
```

Ilustración 65 - Obtención de cada pixel min_image_BN

Con este código se consigue que cada fila de *aux_matrix*, se llenen sus 8 elementos correspondientes a la fila que se está recorriendo.

Consiguiendo así que cada 8 fila recorridas de *filter_image_BN*, cada una de las filas de *aux_matrix* tengan un grupo de 8x8 pixeles. En este código *i* indica el número de columna y *jj* el número de fila de *filter_image_BN*. Por último, *aux_contador* indica el valor de *jj* módulo 8.

- Cada 8 filas recorridas, *aux_matrix* estará completo, por lo que se realizarán las siguientes acciones:
  - Se calcula de la media de cada fila de *aux_matrix*. Comprobando para cada media, lo siguiente:
    - Si la media supera 76, más del 30% de los pixeles son blancos, por lo que se inserta en la posición *contador* de *image_code* el valor 255, que indicará blanco. Se incrementa en 1 *contador*.
    - En caso opuesto, se inserta en la posición *contador* de *image_code* el valor 0, que indicará negro. Se incrementa en 1 *contador*.
  - Una vez calculadas todas las medias de aux_matriz y haber insertado sus valores, se pasa al siguiente conjunto de 8 filas.

Si la imagen cargada en un primero momento está vacía, se devuelve *image_code* sin inicializar.

En el apartado 4.4.2.5 CLASIFICADOR DE IMÁGENES se intentará representará este proceso mediante imágenes, ayudando a su comprensión y mostrar los resultados de realizarlo.

4.4.2.5. Clasificador de imágenes

Este subcomponente se encargará de clasificar las imágenes obtenidas por *Capturador de imágenes*, las cuales, previamente deberá de ser convertidas a un formato comprensible para este subcomponente utilizando el *Procesador de imágenes*. Este subcomponente, solo está compuesto por el archivo *ClasImage.cpp*.

Debido a la infinidad de posibles imágenes distintas que se pueden dar durante la conducción del vehículo, sería imposible clasificarlas todas, para llevar a cabo una conducción autónoma. Por ello, se determinó que el vehículo no podría "*ir por libre*", sino que necesitaría algún elemento que le sirviera de referencia para llevar a cabo esta conducción.

Entonces se decidió, que el vehículo sería conducido sobre un circuito, y este subcomponente, sería el encargado de llevar acabo la tarea de distinguir cada una de las partes del circuito, para que el subcomponente *Servidor* pudiera decidir qué acción realizar dependiendo de dicha clasificación.

Este circuito, estaría formado por dos líneas blancas de 8cm de ancho, con una separación de 24,5cm entre las líneas, todo ello sobre un fondo oscuro. Además, algunas de las formas del circuito, dispondrán de tiras de blancas pequeñas, estas tendrán un tamaño de: 8cm de alto por 2cm de ancho.

Las formas que podría tomar este circuito, y el clasificador debería poder diferenciar, son las siguientes:

- **ATRÁS**

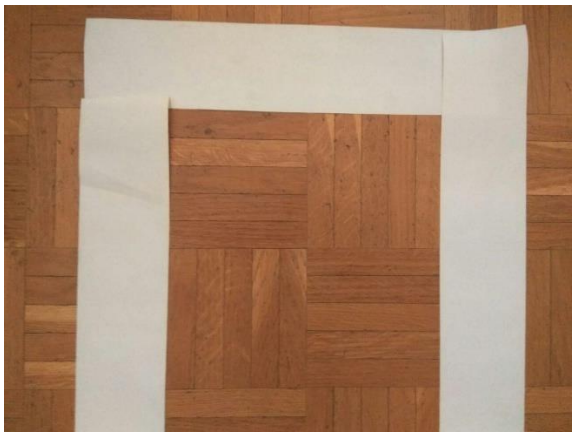

Ilustración 66 - Ejemplo circuito ATRÁS

Esta imagen indicará que el vehículo solo puede ir para atrás. Como este movimiento no está incluido en el conjunto movimientos disponible, se indicará que el único movimiento posible que se puede realizar es parar.

- **DELANTE**

Ilustración 67 - Ejemplo circuito DELANTE

Esta imagen indicará que el vehículo solo puede continua de frente o parar, ya que otro movimiento podría sacarlo del circuito.

- **DERECHA**

Ilustración 68 - Ejemplo circuito DERECHA

Esta imagen indicará que el vehículo deberá girar estrictamente a la derecha, sino se saldrá del circuito.

- **DERECHA-90**

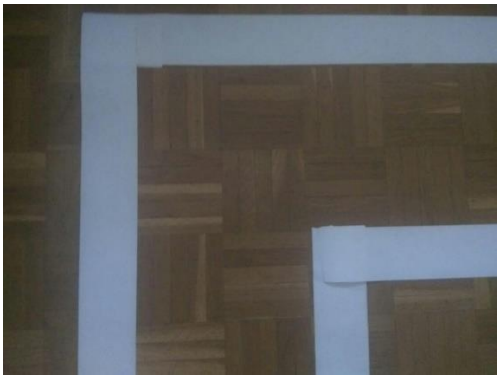

Ilustración 69 - Ejemplo circuito DERECHA-90

Esta imagen indicará que el vehículo puede girar a la derecha o parar, dejando esta decisión al *Servidor*.

- **DERECHA-DELANTE**

Ilustración 70 - Ejemplo circuito DERECHA-DELANTE

Esta imagen indicará que el vehículo puede girar a la derecha, continuar o parar, dejando esta decisión al *Servidor*.

- **IZQUIERDA**

Ilustración 71 - Ejemplo circuito IZQUIERDA

Esta imagen indicará que el vehículo deberá girar estrictamente a la izquierda, sino se saldrá del circuito.

- **IZQUIERDA-90**

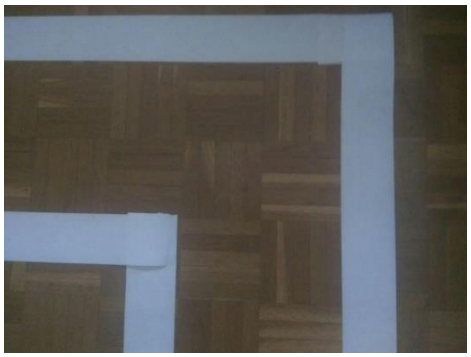

Ilustración 72 - Ejemplo circuito IZQUIERDA-90

Esta imagen indicará que el vehículo puede girar a la izquierda o parar, dejando esta decisión al *Servidor*.

- **IZQUIERDA-DELANTE**

Ilustración 73 - Ejemplo circuito IZQUIERDA-DELANTE

Esta imagen indicará que el vehículo puede girar a la izquierda, parar o avanzar, dejando esta decisión al *Servidor*.

- **IZQUIERDA-DERECHA**

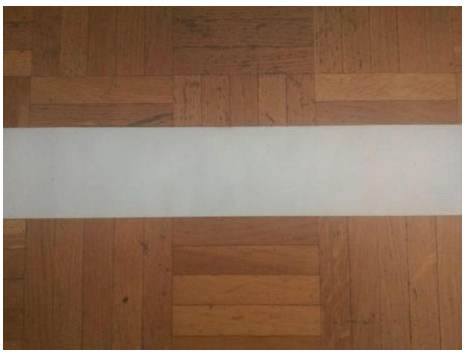

Ilustración 74 - Ejemplo IZQUIERDA-DERECHA

Esta imagen indicará que el vehículo puede girar a la izquierda, derecha o parar, dejando esta decisión al *Servidor*.

- **IZQUIERDA-DERECHA-90**

Ilustración 75 - IZQUIERDA-DERECHA-90

Esta imagen indicará que el vehículo puede girar a la derecha, avanzar o parar, dejando esta decisión al *Servidor*.

- **IZQUIERDA-DERECHA-DELANTE**

Ilustración 76 - Ejemplo circuito IZQUIERDA-DERECHA-DELANTE

Esta imagen indicará que el vehículo puede girar a la derecha, avanzar o parar, dejando esta decisión al *Servidor*.

Las imágenes reales, variaran en mayor o menor medida, respecto a las indicadas, dependiendo del ángulo de la cámara, por lo que el *Clasificador de imágenes* deberá ser entrenado dependiendo del vehículo sobre el que se ejecute el *Servidor*.

Ilustración 77 - Ejemplo circuito DERECHA-90 - ángulo real vehículo

El clasificador no se entrenará con las imágenes en "*bruto*", sino que previamente se procesarán con *Procesador de imágenes*. Quedando el siguiente resultado:

1. Se carga la imagen a procesar, y se le realiza un filtro de blanco y negro. Se utiliza como imagen a procesar, la misma que en ILUSTRACIÓN 77 - EJEMPLO CIRCUITO DERECHA-90 - ÁNGULO REAL VEHÍCULO.

Ilustración 78 - Ejemplo circuito DERECHA-90 - BN

2. Una vez se tiene la imagen en *BN*, se procesa mediante *min_image_BN* de *Procesador de imágenes.*

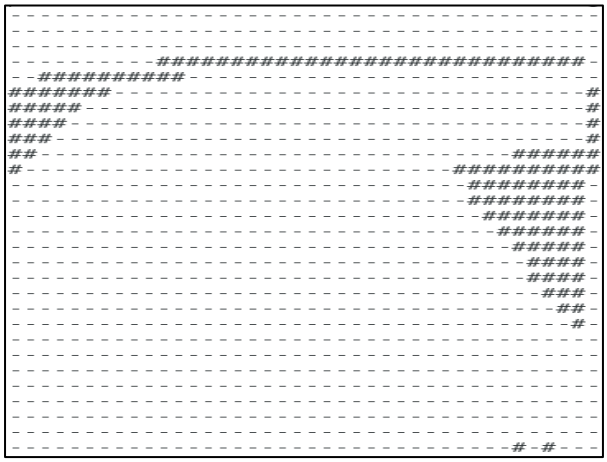

Ilustración 79 - Ejemplo circuito DERECHA-90 - Procesada

3. Dicha imagen procesada, realmente es un *vector* de 1200 posiciones:
    - Los valores “#” corresponden al valor 255, es decir, representan el color blanco.
    - Los valores “-“ corresponden al valor 0, es decir, representan al color negro.

Una vez realizado este inciso, se especifica cual es la implementación realizada para *Clasificador de imágenes* en *ClasImage.cpp*.

Este archivo, implementa la clase *red*, que es la implementación de una red neuronal multicapa en una clase de C++. Por lo tanto, el tipo de clasificador escogido para la aplicación es una red neuronal.

Se decide utilizar una red neuronal como clasificador, debido a que esta se puede entrenar como un clasificador de imágenes no muy complejas, que justo es el caso actual. Además, debido a que *Procesador imágenes* simplifica en gran medida las imágenes de entrada, esta red neuronal funciona con muy buen rendimiento, tardando muy poco tiempo en clasificar las imágenes y obteniendo resultados bastante buenos.

Debido a que el funcionamiento de una red neuronal es conocido y fácil de encontrar, solo se especificarán las funciones que ofrece la red neuronal que han sido implementadas, indicando por encima su funcionamiento:

Constructor por vector

Permite la instanciación del objeto *Red*, indicando por parámetros, un vector con n valores. El primer valor, indicará el número de capas de la red, y el resto de valores, el número de neuronas de cada capa.

Constructor por fichero

Permite la instanciación del objeto, indicando por parámetros la ruta de un fichero que contiene la información de una red neuronal ya entrenada. Este fichero indicará: el número de capas, el número de neuronas en cada capa y todos los pesos de las conexiones entre neuronas.

backpropagation_learning

Función mediante la que aprende la red, utilizando el método propagación hacia atrás. A esta función se le pasa por parámetro lo siguiente:

- Un vector que representa el conjunto de entrenamiento.
- Un vector de respuestas de resultados esperados de clasificación.

Esta función tomará los elementos del conjunto de enteramiento y los intentará clasificar. Una vez clasificado, comparará el resultado obtenido con el esperado. Dependiendo de esta comparación, se actualizarán todo lo pesos de las conexiones de la red.

Se repetirá este proceso de clasificación y actualización de los pesos, hasta que se realice 10000 veces o el error cuadrático sea inferior a 0.0005.

do_output

Función que ejecutará el paso de un elemento a través de la red neuronal. Esta recibe como parámetro, un elemento que se quiera procesar atreves de la red neuronal, y se encarga de llevar a cabo esto.

Devuelve un vector con tantas posiciones como capas de salida. En este vector, por cada una de las posiciones, se devolverá un valor entre 0 y 1, cuanto más cercano sea 1, significará que se ha clasificado con mayor exactitud.

Se dan valores entre 0 y 1, ya que se utiliza la función *sigmoide* como función de activación.

generate_file_network

Función que genera un fichero con todos los datos de una red neuronal. En este fichero se indicará: el número de capas, el número de neuronas por capa y los pesos de todas las conexiones de la red.

4.4.2.6. Servidor

Este subcomponente se encargará se encargará de utilizar al resto al resto de subcomponentes, para llevar acabo el modo de *Control automático* y el modo de *Control manual*. Este se encuentra implementado en un fichero denominado *Servidor.cpp*.

Los atributos globales utilizados por este componente son:

- **recv_env_text**: Array de *char* de 9 posiciones. Se utilizará para almacenar las peticiones enviadas y recibidas, tanto del *Controlador* como de la *Aplicación móvil*.
- **status_cam**: Variable de tipo *bool*. Indicará si la cámara ha sido activada o no. Se inicializa con el valor false.
- **connectionAplicacion**: Objeto de tipo *conAplicacion*. Manejará las comunicaciones con la *Aplicación móvil*.
- **connectionControlador**: Objeto de tipo *conControlador*. Manejará las comunicaciones con el *Controlador*.
- **Neuronal_network:** Objeto de tipo *red*. Se utilizará como clasificador de las imágenes. Se inicializa con un archivo de una red neuronal previamente entrenada. Esta red tiene 3 capas, el número de elementos de cada una es el siguiente:
    - La capa de entrada tiene 1200 neuronas. Esto se debe a que se necesita una neurona por cada pixel de la imagen a clasificar. La función *min_image_BN* de *Procesador de imágenes*, justo devuelve ese tamaño.
    - La capa intermedia tiene 32 neuronas. Este valor no tiene ningún motivo especifico.
    - La capa de salida tiene 11 neuronas. Se debe tener una neurona por cada posible clasificación.
- **Result:** Vector de tipo double. Almacenará el resultado devuelto al clasificar una imagen por *Neuronal_network.*
- **simple_image_BN**: Vector de tipo *uchar*. Almacenará el vector de la imagen que se está tratando.

- **Finish_connection:** Variable de tipo *bool*. Indicará, en el modo de controla automático, si se ha finalizado la conexión con el cliente o no.

Las funciones de las que dispone este subcomponente son:

Main

Lo primero que se realiza en esta función, es establecer conexión con el *Controlador*. Para ello, se inicializa *connectionControlador* pasándole como argumento de entrada "/dev/ttyACM0". Una vez realizada esta función, se comprueba si se ha establecido la conexión con el *Controlador*. Si no está establecida, se imprime por pantalla y se cierra el programa.

Una vez conectado al *Controlador*, inicializa *connectionAplicacion,* pasándole como parámetro de entrada la IP obtenida con la función *ip_localhost*. Esto dará paso a que los clientes de la *Aplicación Móvil* puedan conectarse.

Una vez ya están abiertos los canales de comunicación, esta función entra en un bucle sin condición de salida. En el cual se realiza lo siguiente:

- Se esperan la conexión de clientes de forma indefinida, mediante la función *listenClient* de *connectionAplicacion*.
- Cuando se establece una conexión con un cliente, se espera de forma indefinida a recibir una petición de modo mediante la función *recibir* de *connectionAplicacion*. Dependiendo de la petición que se reciba se llevan a cabo las siguientes acciones:
    - Si la petición es "MOD: MAN\n", significará que es una petición de modo de *Control manual*. Se responde utilizando *enviar* de *connectionAplicacion* la respuesta "MOD: OK\n" y se llama a la *función_tratamiento_pet_manual*.
    - Si la petición es "MOD: AUT\n", significará que es una petición de modo de *Control automático*. Se responde utilizando *enviar* de *connectionAplicacion* la respuesta "MOD: OK\n" y se llama a la *función_tratamiento_pet_automatico.*
- Una vez se ha finalizado las funciones del tratamiento del modo manual o el automático. Se cierra la conexión con el cliente mediante *closeClient* de *connectionAplicacion*.

Si por algún casual, se sale del bucle, se realiza *closeServer* y *closeArduino* de *connectionAplicacion* y *connectionControlador* respectivamente.

tratamiento_pet_manual

Función que se encarga de las peticiones del tratamiento de peticiones en el modo de *Control manual*.

Lo primero que realiza esta función, es imprimir por pantalla que se ha entrado en el modo de *Control manual*. Una vez hecho esto, se declara una variable booleana *finish_connection* que se utilizará para indicar si se ha finalizado la conexión con el cliente que actualmente realiza peticiones.

Una vez hecho esto, se entra en un bucle cuya condición de salida es que *finish_connection* indique que se ha finalizado la conexión. En este bucle se realiza lo siguiente:

- Se espera la recepción de una petición de forma indefinida, mediante la función *recibir* de *connectionAplicaction*.
- Una vez recibida una petición, se imprime por pantalla y se comprueba que petición es:
    - Si es "CAM: ON\", significara una activación de la cámara. Por lo primero que se hace es comprobar *status_cam* para observar si la cámara esta activa. Dependiendo de esto se realiza:
        - Si ya está activa, se copia en *recv_env_text* "CAM: OK\n"
        - Si no está activa, se lanza un proceso hijo, que se bifurca del flujo de ejecución del padre:
            - El hijo, se utiliza para lanzar el proceso *Motion* para la transferencia de imágenes. Para ello, se ejecuta la función *execl* con el comando "motion -n"
            - El padre, indica en *rec_env_text* "CAM: OK\n" y cambia status_cam a true.
    - Si es "CAM: OFF\n", significará una desactivación de la cámara. Por lo primero que se hace es comprobar *status_cam* para observar si la cámara esta activa. Dependiendo de esto se realiza:
        - Si ya está desactivada, se copia en *recv_env_text* "CAM: OK\n"
        - Si esta activada, se ejecuta *kill* con el *pid* del proceso hijo que estaba ejecutando *Motion*. Después, se cambia *status_cam* a false y se copia *recv_env_text* "CAM: OK\n"
    - Si es "MOD: OFF\n" significará una finalización de modo. En este caso, se copia en *recv_env_text* "MOD: OK\n y se cambia *finish_connection* a true.
    - Si la petición no coincide con ninguna de las anteriores, se envía al *Controlador*, mediante *enviar_arduino*, y se recibe la respuesta, me *respuesta_arduino*. Las dos funciones son de *ConnectionControlador*.
- Una vez tratada la petición, se imprime la respuesta que encuentra en *recv_env_text* y se envía, mediante *enviar* de *connectionAplication*.

Si sucede cualquier error al enviar o recibir elementos al cliente, se finalizará la función. En el caso de que *finish_conection* tome el valor true, se finalizará el bucle y por consiguiente la función.

automatic_mode_thread_slave

Función ejecutada por un hilo para recoger peticiones realizadas por el cliente. Este hilo se lanzará en *tratamiento_pet_automatico.*

Lo único que realiza la función es esperar constantemente a peticiones del cliente, mediante *recibir* de *connectionAplication*. Una vez hecho esto, se comprueba que petición se ha recibido, realizando las siguientes acciones:

- Si la petición es "CAM: OFF\n", se envía "CAM: OK\n" al cliente y se cambia *finish_connection* a true.
- En caso contrario, se envía "MSG: ERR\n" al cliente.

Se realiza esta comprobación, ya que, en *Control automático*, solo se realizará el envío de peticiones de desactivación de la cámara desde la *Aplicación móvil*, si se está finalizando la conexión con el *Servidor*.

tratamiento_pet_automatico

Función que se encargará de llevar acabo la conducción autónoma del vehículo dependiendo de las imágenes recibidas desde la cámara.

Nada más entrar a esta función, lo primero que se realiza, es una impresión por pantalla indicando el modo de control escogido.

Una vez hecho esto, se empieza el proceso de recepción de peticiones previo a comenzar la conducción automática. En este proceso, se realizan las siguientes acciones en el orden en el que se citan:

- Se espera una petición de activación de cámara, mediante *recibir* de *connectionAplication*. Si se recibe la petición, se realiza lo siguiente:
  - Se lanza un proceso hijo que ejecuta *Motion*, con el fichero de configuración correspondiente a este modo.
  - El proceso padre envía al cliente "CAM: OK\n"
- Se espera una petición de especificación de número de movimientos de un trayecto, mediante *recibir* de *connectionAplication*. Si se recibe la petición, se realiza las siguientes acciones en orden:
  - Se espera una petición de número de movimientos. Si se recibe, se inicializa un vector con estos movimientos y se envía al cliente "MOD: OK\n"
  - Se esperan tantas peticiones, como número de movimientos se hayan especificado en la petición anterior. Se comprueba que cada petición corresponde con un movimiento del trayecto y se almacena en el vector antes mencionado. Además, por cada movimiento se enviará al cliente "MOD: OK\n"

Si durante el proceso de recepción de peticiones, se produce un error o la petición recibida no es la esperada, se enviará al cliente "MSG: ERR\n"

Una vez se tiene la información del trayecto, lo primero que se hace es mandar una petición al *Controlador* de tipo "CAM: CEN\n", por si la cámara no está centrada.

Una vez hecho esto, se lanza el hilo que ejecuta *automatic_mode_thread_slave.*

Preparados todos los elementos necesarios para la conducción autónoma, se entra en un bucle, cuya condición de salida es que *finish_connection* valga true o se hayan realizado todos los movimientos. En este bucle, se realizan las siguientes acciones en orden:

- Se obtiene la imagen más actual posible que haya almacenado *Motion* y se convierte en el formato correspondiente con *min_image_BN*. Para ello, se utiliza la función *obtain_image_name* del *Procesador de Imágenes*. Se comprueba desde 24 a 0 las imágenes creadas en el segundo actual, ya que como máximo se habrán generado 24 por el *framerate* escogido.
- Se comprueba que dicha imagen no se haya procesado anteriormente. Si no es así, se vuelve a intentar cargar la imagen más actual.
- Se clasifica la imagen actual, mediante *do_Output* de neuronal *network*.
- Del *vector* devuelto por *do_Output*, se obtiene cuál de las 11 posiciones es la que tiene el valor más alto, y, por lo tanto, se haya clasificado de forma más precisa. También se comprueba si dicho valor supera 0.5, si no lo supera se descarta y se vuelve a cargar otra imagen.
- Si el valor supera 0.5, se imprime por pantalla la imagen que se ha clasificado, y se añade su posición dentro del vector devuelto por *do_Output*, dentro un vector de almacenará las ultimas 5 imágenes clasificadas.
- Se comprueba si las últimas 5 imágenes son clasificadas igual, si es así, se indica mediante una variable denominada *last_five_moves* la posición de las 5 imágenes dentro del vector devuelto por *do_Output*. En caso contrario, toma el valor -1.
- Una vez hecho todo esto, se comprueba cual es la posición de la última imagen dentro del vector devuelto por *do_Output.*
    - **Si su valor es 2 o 5:** Significa **IZQUIERDA** o **DERECHA**, por lo tanto, que se el vehículo se está saliendo del circuito. Por ello, independientemente del valor de *last_five_moves* y el movimiento a realizar, se envía una petición al *Controlador* con *enviar_arduino_dir*, moviendo el vehículo en la dirección que le centre en el circuito.
    - **Si su valor es 8**: Significa **IZQUIERDA-DERECHA**, y *last_five_moves* no es -1. Se comprueba cuales es el valor de *last_five_moves:*
        - **Si es 9, IZQUIERDA-DERECHA-90**. Se comprueba el movimiento actual a realizar:
            - Si es girar a la izquierda o derecha, se comienza a girar el vehículo hasta realizar un giro cercano a los

90º en dichos sentidos. Se envía el estado del movimiento “MOV: OK\n”.

- ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente “MOV: OK\n”.
- ❖ Si el movimiento es ir hacia delante, se detiene el vehículo. Se envía el estado al cliente “MOV: NO\n”.

- **Si es 6**: significa **IZQUIERDA-90**. Se comprueba el movimiento actual a realizar:
  - ❖ Si es girar a la izquierda, se comienza a girar el vehículo hasta realizar un giro cercano a los 90º en dicho sentido. Se envía el estado del movimiento “MOV: OK\n”.
  - ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente “MOV: OK\n”.
  - ❖ Si el movimiento es ir hacia delante o derecha, se hace un giro de cercano a 90º a la izquierda. Se envía el estado al cliente “MOV: NO\n”.
- **Si es 3**: significa **DERECHA-90**. Se comprueba el movimiento actual a realizar:
  - ❖ Si es girar a la derecha, se comienza a girar el vehículo hasta realizar un giro cercano a los 90º en dicho sentido. Se envía el estado del movimiento “MOV: OK\n”.
  - ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente “MOV: OK\n”.
  - ❖ Si el movimiento es ir hacia delante o izquierda, se hace un giro de cercano a 90º a la derecha. Se envía el estado al cliente “MOV: NO\n”.

- **Si su valor es 8**: Significa **IZQUIERDA-DERECHA**, y *last_five_moves* es -1. Se detiene el vehículo.
- **Si su valor es 1**: Significa **DELANTE**, y *last_five_moves* no es -1. Se comprueba cuales es el valor de *last_five_moves:*
  - **Si es 4, DERECHA-DELANTE**. Se comprueba el movimiento actual a realizar:
    - ❖ Si es ir de frente o derecha, se avanza el vehículo o se comienza a girar el vehículo hasta realizar un giro cercano a los 90º en dicho sentido. Se envía el estado del movimiento “MOV: OK\n”.
    - ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente “MOV: OK\n”.
    - ❖ Si el movimiento es ir hacia izquierda, continua de frente. Se envía el estado al cliente “MOV: NO\n”.
  - **Si es 7**: significa **IZQUIERDA-DELANTE**. Se comprueba el movimiento actual a realizar:

- ❖ Si es ir de frente o izquierda, se avanza el vehículo o se comienza a girar el vehículo hasta realizar un giro cercano a los 90º en dicho sentido. Se envía el estado del movimiento "MOV: OK\n"
- ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente "MOV: OK\n"
- ❖ Si el movimiento es ir hacia derecha, hace avanzar al vehículo. Se envía el estado al cliente "MOV: NO\n"

- **Si es 3**: significa **IZQUIERDA-DERECHA-DELANTE**. Se comprueba el movimiento actual a realizar:
  - ❖ Si es girar a la derecha o izquierda, se comienza a girar el vehículo hasta realizar un giro cercano a los 90º en dichos sentidos. Se envía el estado del movimiento "MOV: OK\n"
  - ❖ Si el movimiento es parar, se detiene el vehículo. Se envía el estado al cliente "MOV: OK\n"
  - ❖ Si el movimiento es ir hacia delante, se hace avanzar al vehículo. Se envía el estado al cliente "MOV: OK\n".

- **Si su valor es 1**: Significa **DELANTE**, y *last_five_moves* es -1. Se obliga avanzar al vehículo.
- **En cualquier otro caso**, se obliga a avanzar el vehículo.

- Si la imagen no se reconoce, se obliga a avanzar el vehículo, ya que puede ser una imagen no reconocida en dicho instante.

Para la realización de movimientos de avanzar y parar se usa *enviar_arduino_mtr* de *connectionControlador*. En el caso de girar se utiliza *enviar_arduino_dir* de *connectionControlador*.

Una vez finalizado el trayecto o se haya modificado el valor de *finish_connection.* Se llevan a cabo las siguientes acciones:

- Se imprime por pantalla que se ha finalizado el trayecto.
- Se espera la finalización del hilo receptor de peticiones.
- Se envía "MOD: OK\n" al cliente.
- Se envía *kill* al proceso hijo encargado de la cámara.
- Se termina la función.

4.4.3. Controlador

En este apartado, se hablará de la implementación realizada para el componente *Controlador*, el cual, como ya se ha comentado con anterioridad, es un *sketch* de Arduino escrito en C/C++. Este se ejecutará sobre un Arduino UNO con un *Shield* Adafruit Motor/Stepper/Servo Shield v1.2, que se encargarán de interactuar con los actuadores y sensores del vehículo, excluyendo a la cámara.

El circuito eléctrico, que gobierna el hardware que ejecuta este componente, es el siguiente:

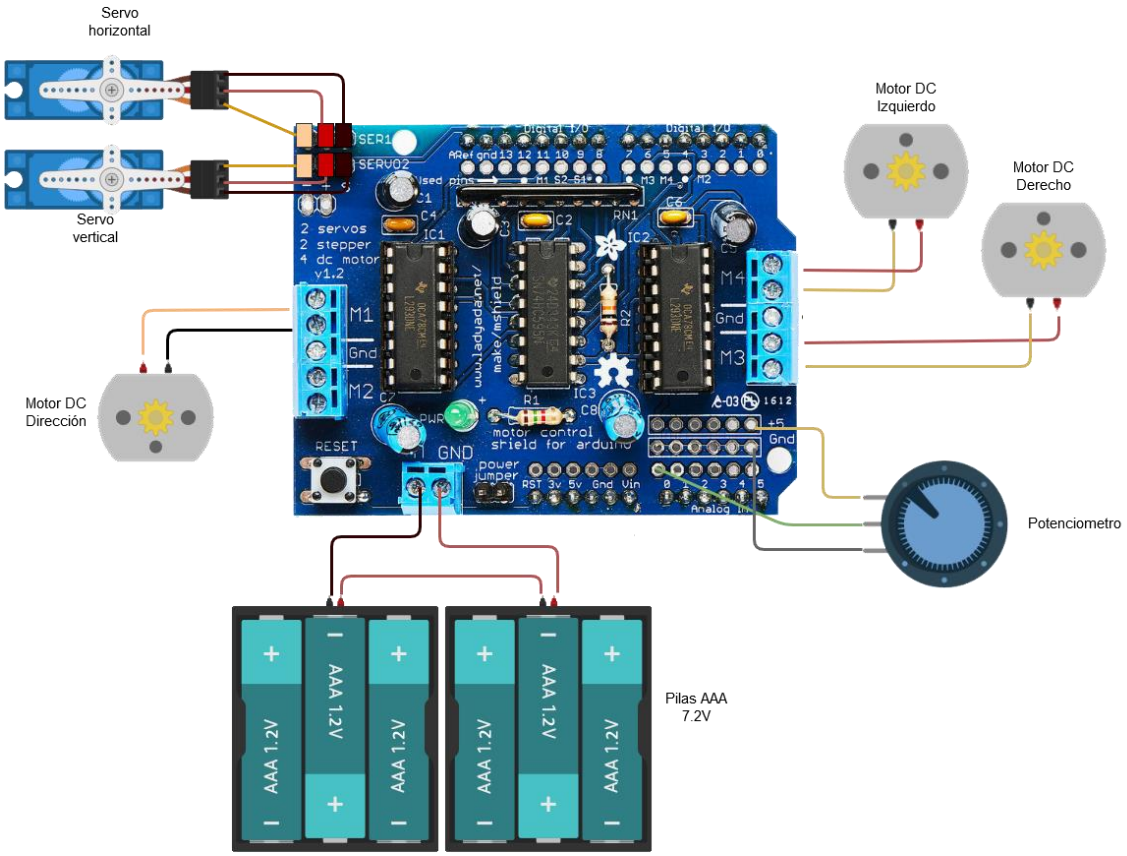

Ilustración 80 - Circuito eléctrico vehículo

- **Arduino UNO:** Este elemento no se muestra, debido a que se encuentra oculto bajo el *Shield*. Todos sus pines se encuentran conectados al *Shield*.
- **Servo horizonta**l: Se encargará de llevar a cabo los movimientos de giro de la cámara hacia arriba y hacia abajo. Está conectado a *Servo1* del *Shield*, cuyo pin de señal, es el pin digital 9 del Arduino UNO.
- **Servo vertical**: Se encargará de llevar a cabo los movimientos de giro de la cámara hacia la derecha y la izquierda. Está conectado a *Servo2* del *Shield*, cuyo pin de señal, es el pin digital 10 del Arduino UNO.
- **Motor DC dirección**: Se encargará de realizar los giros de la dirección a la izquierda, a la derecha y centrar la dirección. Estará conectado a *M1* del *Shield*.

- **Pilas 7.2V AAA:** Se encargará de realizar la alimentación externa del vehículo. Este componente son 6 pilas recargables AAA de 1.2V en serie.
- **Potenciometro**: Está conectado al eje del *Motor de dirección*, por lo que se modificará su valor con el giro de la dirección del vehículo. Por lo tanto:
    - Dará el valor 350Ω o menor, cuando la dirección se encuentre totalmente girada a la izquierda.
    - Dará el valor 675Ω o mayor, si la dirección se encuentra totalmente girada completamente a la derecha.

  Por último, este potenciómetro está conectado a la entrada A0 del *Shield* para medir su valor, esta corresponde con la entrada A0 del Arduino UNO.
- **Motor DC Izquierdo:** Es el motor de propulsión que se encuentra posicionado a la izquierda del vehículo. Este esta conectado a *M4* del *Shield*.
- **Motor DC derecho:** Es el motor de propulsión que se encuentra posicionado a la derecha del vehículo. Este esta conectado a *M3* del *Shield*.

Este componente utiliza librerías externas desarrolladas por terceros. A continuación, se indican cuales son y porque se utilizan:

Servo

[B47] Es una librería desarrollada por Arduino. Ofrece una serie de funcionalidades para el control preciso de servomotores conectados a una placa Arduino. Se decide utilizar esta librería por los siguientes motivos:

- El circuito del vehículo dispone de dos servomotores, que pueden ser manejados fácilmente y de forma precisa, mediante esta librería.
- Dispone el objeto *Servo,* que permite asociarle un pin de señal al que se conecta un servomotor.
- El objeto *Servo* dispone de una función denominada *write*, a la que se le pasa como argumento de entrada un valor entre 0 a 180. Esta se encargará de enviar, a través del pin de señal asociado al objeto *Servo*, el valor indicado como argumento de entrada de esta función.
- Los servomotores utilizados por el vehículo aceptan valores entre 0 a 180.

AFmotor

[B48] Es una librería desarrollada por AdaFruit. Ofrece todas las funcionalidades necesarias para la utilización del *Shield* Adafruit Motor/Stepper/Servo Shield v1.2. Se decide utilizar por los siguientes motivos:

- El *Controlador* utiliza este *shield* para el control de todos los actuadores del vehículo, así como, para obtener las medidas del potenciómetro.
- Dispone del objeto *AF_DCMotor*, que permite el control de motores DC conectados a las entradas *M1*, *M2*, *M3* y *M4*, del *Shield*. Al inicializar este objeto, se indica un valor del 1 al 4, que corresponde con la entrada *MX* al que está conectado el motor que se quiere controlar.

- El objeto *AFDC_Motor*, mediante la función *setSpeed*, permite especificar un valor de velocidad entre 0 a 255. Donde 0, significa que no se envía señal al motor, y 255, que se suministra de forma continua la señal. Si se asigna un valor entre 1 a 254, la función se encargará de realizar las configuraciones necesarias para llevar a cabo una señal *PWM* que regule la velocidad del motor.
- El objeto *AFDC_Motor*, mediante la función *run*, permite indicar el sentido de la corriente que pasa atreves del motor DC asociado. En dicha función, se dispone de tres posibles sentidos de la corriente:
    - RELEASE: No se produce corriente en las entradas de *MX*.
    - FORWARD: La corriente sale de la primera entrada de *MX* y la segunda entrada pasa a convertirse en una tierra.
    - BACKWARD: La corriente saldrá de la segunda entrada de *MX* y la primera entrada se convertirá en una tierra.

Este componente no tiene subcomponentes. Por lo que, a continuación, se definirá su implementación en este mismo apartado.

Las variables globales y constantes utilizadas por el *sketch* son la siguientes:

- **AUX_TICK_SERVO:** Constante que indicará cuantas veces se tiene que ejecutar la función *loop* para que se aumente o disminuya un grado en alguno de los servomotores.
- **serv_vertical y serv_horizontal:** Objetos de tipo *Servo*. Se utilizarán para manejar los servos del vehículo.
- **pos_vertical y pos_horizontal:** Variables de tipo *int.* Almacenarán el ángulo de los servomotores. Sus valores iniciales será 90.
- **ver_orientacion y hor_orientacion**: Variables de tipo *int*. Indicarán la dirección de giro de los servomotores. Tomarán el valor 0 si está parado, 1 si está aumentando su angulo y 2 si lo está disminuyéndolo. Sus valores iniciales serán 0.
- **ver_change y hor_change**: Variables de tipo booleano. Indicarán si ha habido una petición que pueda cambar el sentido de giro de los servomotores.
- **aux_horizontal y aux_vertical:** Variables de tipo *int*. Se utilizarán como contadores de cuantas veces que se ejecuta la función *loop*, si se ha realizado una petición de movimiento de alguno de los servomotores. Sus valores iniciales serán 0.
- **direccion:** Objeto de tipo *AF_DCMotor*. Se le inicializará con el valor 1, debido a que el motor de la dirección está conectado a M1.
- **dir_orientacion:** Variable de tipo *int*. Indica la dirección a la que se está girando la dirección, tomando el valor 0 si esta parada, 1 si está girando a la izquierda, 2 si está girando a la derecha y 3 si se está centrando.
- **dir_pin_pot:** Variable de tipo *int*. Indica el pin analógico desde el que se leerá el valor del potenciómetro.

- **dir_val_pot:** variable de tipo *int*. Almacenará el valor del potenciómetro.
- **dir_change y mtr_change:** Variables de tipo booleano. Indicará si ha habido una petición que cambie el sentido de giro del motor de la dirección o de los motores de propulsión.
- **motor1:** Objeto de tipo *AF_DCMotor*. Se le inicializará con el valor 3, debido a que el motor de propulsión izquierdo está conectado a M3.
- **motor2** Objeto de tipo *AF_DCMotor*. Se le inicializará con el valor 3, debido a que el motor de propulsión derecho está conectado a M4.
- **motor_orientacion:** Variable de tipo *int*. Indica el sentido de giro de los motores de propulsión, tomará el valor 0 si están parados, 1 si están avanzando y 2 si están retrocediendo.

Las funciones de las que dispone este componente son las siguientes:

Setup

Función de la que disponen los *sketch's*, la cual, es llamada la primera una vez cargado el *Sketch*. En este caso, las acciones que se realizan en esta función son:

- Se inicializa el puerto serial a 9600 *baudios*, ya que será necesario para realizar la comunicación con el *Servidor*.
- Se asocia a *serv_vertical* el pin 9 y a *serv_horizontal* el pin 10. Una vez asociados, se llama a la función *write* de los dos y se le pasa como parámetro de entrada *pos_vertical* y *pos_horizontal*.
- Se llama a las funciones *setSpeed* y *run* de todos los motores dc conectados al *shield*, pasado como parámetros de entrada *RELEASE* y *200*.

Loop

Función que de la que disponen los *sketch's*, la cual, una vez realizado la función *setup*, se realiza en bucle hasta que se desconecte el microcontrolador. En este caso, a esta función, se le añade el código para que llame a las funciones: *comunication*, *control_motor*, *control_direccion*, *control_serv_vertical* y *control_serv_horizontal*.

comunication

Función que se encarga de recibir las peticiones del *Servidor***.** Para llevar acabo esto, lo primero que hace esta función, es instanciar dos *arrays* de tipo *char* de 10 posiciones, denominados *respuesta* y *peticion*.

Una vez instanciados los *arrays*, se comprueba si en el puerto serial hay al menos 9 bytes, ya que las peticiones realizadas al *Controlador* deben de tener este tamaño. Si no se dispone de al menos 9 bytes, se finaliza la función.

Si se dispone de 9 bytes, se recogen y almacenan en *peticion*. Una vez recogida una petición, se comprueba que petición ha sido la realizada, comparando *peticion* con todas las peticiones que se puede realizar al *Controlador*.

Dependiendo de la petición, se modifican las variables globales correspondientes para indicar que se ha realizado una petición que cambia el estado de los actuadores correspondientes, y el movimiento que deberían hacer ahora.

La única petición con comportamiento distinto es la de centrar la cámara, la cual, se realizará directamente sobre esta función. Para ello, realiza las siguientes acciones en el orden en que se citan:

- Salvo las variables *Servo*, se establecen los valores iniciales de todas las variables globales utilizadas para controlar los servomotores.
- Se llama a la función *write* de los dos servomotores, pasándoles como parámetro de entrada *pos_horizontal* y *pos_vertical*.

Una vez tratada la petición, se almacena en *respuesta,* la respuesta que indica que la petición se ha realizado de forma correcta. Si la petición almacenada en *peticion*, no es reconocida, se almacena en *respuesta,* "MSG: ERR\n" que indica que ha sucedido un error. Por último, se envía *respuesta* por el puerto serial y se finaliza la función.

control_motor

Función encargada de gestionar el movimiento de los motores de propulsión cuando se quiere avanzar o retroceder. Lo primero que realiza esta función es comprobar el valor de *mtr_change*, que indicará si se ha producido un cambio en el movimiento de los motores.

Si no se ha producido un cambio, no se realizan más acciones y se finaliza la función.

Si se ha producido un cambio, se comprueba el valor de *motor_orientation,* dependiendo de su valor se realiza:

- Si tiene el valor 0, indicará que se ha realizado una petición de parada de los motores, por lo que se llama a la función *run* de *motor1* y *motor2*, pasándole como argumento de entrada *RELEASE*.
- Si tiene el valor 1, indicará se ha realizado una petición para avanzar, por lo que se llama a la función *run* y *setSpeed* de *motor1* y *motor2*, pasándoles como parámetros de entrada *FORWARD* y 200.
- Si tiene el valor 1, indicará se ha realizado una petición para retroceder, por lo que se llama a la función *run* y *setSpeed* de *motor1* y *motor2*, pasándoles como parámetros de entrada *BACKWARD* y 200.
- Se cambia el valor *mtr_change* a false, que indicará que no hay cambios de movimiento en los motores.

control_direccion

Función que gestiona los movimientos del motor de la dirección y los motores de propulsión cuando se realiza una petición de giro del vehículo. Lo primero que realiza esta función es comprobar *dir_change*, que indicará si se ha producido un cambio en los movimientos de giro del vehículo.

Si se ha producido un cambio, se comprueba el valor de *dir_orientation,* dependiendo de su valor se realiza:

- Si tiene el valor 0, indicará que se ha realizado una petición de parar de girar, por lo que se llama a la función *run* de *dirección*, *motor1*y *motor2*, pasándole como parámetro de entrada *RELEASE*.
- Si tiene el valor 1, indicará que se quiere realizar un giro a la izquierda, por lo que se realizan las siguientes acciones:
    - Se lee el valor del potenciómetro y se almacena en *dir_val_pot.*
    - Se comprueba el valor de *dir_val_pot*. Si supera 350, indicará que la dirección no está completamente a la izquierda, por ellos, se llama a la función *run* y *setSpeed* de *direccion*, pasándoles como parámetros de entrada *FORWARD* y 255, lo que hará girar la dirección a la izquierda. Si no supera 350, indicará que la dirección ya está girada a la izquierda, por lo que se llama a *run de direccion*, pasándole como parámetros de entrada *RELEASE*, lo que detendrá el motor de la dirección si este estaba activado.
    - Se llama a la función *run* y *setSpeed* de *motor2*, pasándole como parámetros de entrada *FORWARD* y 255, lo que hará que gire el motor derecho hacia delante, ayudando a realizar el giro.
    - Se llama a la función *run* de *motor1*, pasándole como parámetro de entrada RELEASE, lo que detendrá el motor izquierdo.
- Si tiene el valor 2, indicará que se quiere realizar un giro a la derecha, por lo que se realiza un proceso similar al del caso anterior. Pero, en este caso, comprobando que *dir_val_pot* no supera 675, girando *direccion* a la derecha en ese caso, y en caso contrario, se detiene *direccion*. Además, en este caso el motor que comenzará a girar será *motor1*, el izquierdo, y el que se detendrá será *motor2*, el derecho.
- Se modifica *dir_change* a false, que indicará no hay cambios en los movimientos de giro del vehículo, y se finaliza la función.

Si no se ha producido un cambio en los movimientos de giro del vehículo, se comprueba el valor de *dir_orientation,* dependiendo de su valor se realiza:

- Si tiene el valor 3, indicará que el vehículo está en un estado de centrado constante de la dirección, hasta que se especifique otro movimiento. Para llevar esto acabo se realiza lo siguiente:
    - Se lee el valor del potenciómetro y se almacena en *dir_val_pot,*
    - Se comprueba el valor de *dir_val_pot*. Dependiendo de su valor se realiza:
        - Si tiene un valor entre 475 y 525, indicará que la dirección está centrada. Por lo que se detendrá *direccion*, llamando a su función *run* y pasándole el valor RELEASE.
        - En caso de que su valor no esté dentro de este intervalo, se comprueba si se supera o está por debajo. Si está por

debajo, indicará que la dirección esta girada a la izquierda, por lo que se llama *run* y *setSpeed* de *direccion*, pasándoles como parámetros de entrada *BACKWARD* y 75. Si está por debajo, se realizan las acciones opuestas para girar la dirección a derecha, pero con la misma velocidad. Se decide utilizar velocidades tan bajas, para que al Arduino le dé tiempo a leer el valor del potenciómetro.

- Si tiene otro valor, se finaliza la función.

control_serv_vertical y control_sev_horizontal

Son las funciones encargadas de llevar acabo los movimientos de los servomotores. Solo se especificará *control_serv_vertical*, ya que *control_serv_horizontal* realiza el mismo flujo de ejecución, pero con las variables correspondientes a su control.

Lo primero que realiza esta función, es comprobar *ver_change*, que indicará si se ha producido un cambio en los movimientos del servomotor vertical.

Si no se han producido cambios, se finaliza la ejecución de esta función.

Si se ha producido un cambio, se comprueba el valor de *ver_orientation.* *D*ependiendo de su valor se realizan las siguientes acciones:

- Si tiene el valor 0, significará que se ha enviado la petición de parada del servo. En este caso, solo cambia la variable *ver_change* a false, indicando que no hay cambios en los movimientos del servomotor vertical.
- Si tiene el valor 1, significará que quiere girarse la cámara hacia arriba, es decir, aumentar el ángulo del servo. Por lo que se realizarán las siguientes acciones:
    - Se comprueba mediante *pos_vertical* que el ángulo del servo es menor que 180 y que *aux_vertical* es igual a *AUX_TICK_SERVO*:
        - Si se cumplen las dos condiciones, se cambia el valor de *aux_vertical* a 0, se incrementa en 1 *pos_vertical*, y se llama a la función *write* de *serv_vertical* pasándole como parámetro de entrada *pos_vertical.* Con esto, se habrá aumentado el grado de apertura del servo en un grado. Por último, se finaliza la función.
        - Si no se cumplen, se incrementa en 1 *aux_vertical* y se finaliza la función.
- Si tiene el valor 2, significará que quiere girarse la cámara hacia abajo, es decir, disminuir el ángulo del servo. Por lo que se realizarán las siguientes acciones:
    - Se comprueba mediante *pos_vertical* que el ángulo del servo es mayor que 0 y que *aux_vertical* es igual a *AUX_TICK_SERVO*:
        - Si se cumplen las dos condiciones, se cambia el valor de *aux_vertical* a 0, se decrementa en 1 *pos_vertical*, y se llama

a la función *write* de *serv_vertical* pasándole como parámetro de entrada *pos_vertical,* disminuyendo así el grado de apertura del servo en un grado. Por último, se finaliza la función.

- Si no se cumplen, se incrementa en 1 *aux_vertical* y se finaliza la función.

En esta función, se compara *aux_vertical* con *AUX_TICK_SERVO*, para que al menos se deba de realizar *AUX_TICK_SERVO* veces la función *loop,* antes de aumentar o disminuir en un el ángulo el servomotor. Si no se hiciera esto, el servomotor variaría su ángulo muy rápido, lo que dificultaría en gran medida su control.

5. Marco regulador e impacto socioeconómico

En este apartado, se hablará del marco regulador e impacto socioeconómico que afectan al proyecto.

El primero, especificará todas aquellas legislaciones, convenios, etc. Que regulan el resultado final del proyecto.

El segundo, hablará de la influencia que tendrá en la sociedad y en la industria. En general, en influencia y cambios en hábitos de la sociedad.

5.1. Marco regulador

En este apartado, se hablarán de todos aquellos elementos regulan este tipo de proyectos.

Actualmente, en España no existe regulación con respecto al uso de vehículo autónomos. Por el momento, solo está establecido un marco para realizar la solicitud de pruebas de conducción en la carretera [B54].

Por ello, en este apartado solo se tendrán en cuenta el marco regulador que afecta a la aplicación como tal, sin tener en cuenta que esta controla un automóvil.

5.1.1. Legislación aplicable

La aplicación deberá de cumplir la LOPD [B55] (Ley orgánica de protección de datos personales). La cual, se encarga de garantizar que la seguridad de la información del usuario personal de un usuario o aquella que pueda afectar a la integridad del mismo, si es revelada.

5.1.2. Cumplimiento de propiedad intelectual

La aplicación deberá de respetar las licencias de uso de cada uno de los componentes que utiliza, los cuales, han sido desarrollados por terceros, ya que, en determinados casos, estos no se podrán utilizar si la aplicación va a ser comercializada:

- **OpenCV:** Dispone de una licencia BSD, la cual permite la creación de productos derivados de esta, pero requirieren la adecuada atribución de autoría. Además, permite su redistribución y modificación.
- **Serial, Servo, Motion y WirinPI:** Disponen de una licencia GNU GPL, es un tipo de licencia que permite la modificación y comercialización del código que licencias. Pero, es necesario liberar el código que las utiliza, siendo el también el código resultante de su uso GNU GPL.
- **AFmotor:** Su código es de dominio público. Esto significa, que se puede utilizar con total libertad.

5.2. Impacto socioeconómico

[B58] **El vehículo autónomo, entre lo digital y lo industrial**

Desde una perspectiva científico-tecnológico, él vehículo autónomo está hibridando los procesos industriales más tradicionales de una de las cadenas de valor más complejas (la fabricación de un vehículo) con el desarrollo de software inteligente y algoritmos de inteligencia artificial.

Así, son muchos los que creen que los vehículos autónomos serán máquinas realmente complejas con importantes dotaciones tecnológicas, tanto industriales como digitales.

Pero el impacto del vehículo autónomo va más allá.

Afecta a otras dimensiones de análisis, como pueden ser la económica, industrial, social, ética, legal, etc. La visión de la incorporación tecnológica que un vehículo sufrirá, sino también en cuanto al papel de las tecnológicas en el devenir del sector, y su complementariedad con los fabricantes de vehículos más tradicionales.

El vehículo autónomo ofrecerá la posibilidad de pensar en nuevos productos o servicios, así como nuevos modelos de negocio. Es posible que el vehículo autónomo pueda redefinir y crear nuevas oportunidades, en las complejas cadenas de valor que tiene el sector de la automoción.

En zonas donde se cuentan las plantas de fabricación, y donde se cuentan con numerosas empresas que trabajan en diferentes niveles y partes del proceso de fabricación de un vehículo resulta de interés hablar en clave industrial de cadenas de valor.

Otro tema importante es la dimensión regulatoria y normativa.

Los nuevos retos tecnológicos provocan que las leyes deban adaptarse. En este caso, al hablar de movilidad y uso ciudadano de un vehículo, el código de circulación y las normas de tráfico. La DGT, ya está trabajando en un proyecto para que la convivencia con los vehículos autónomos sea posible.

Por último, está la dimensión social. Desde una mirada del impacto que tiene en la sociedad en su conjunto, el vehículo autónomo despierta varias inquietudes y reflexiones alrededor de la redefinición de la responsabilidad civil, la robotización de profesiones (taxistas, transportistas, etc.), automatización de procesos manuales, etc. que nos deben de llevar a un estudio ético, de la utilización del mismo.

Conclusión

Cada vez estamos más cerca de tener coches autónomos funcionales, que conduzcan completamente solos mientras nosotros nos relajamos. Esto da lugar a debates sobre ética y también sobre legislación, pero a su vez, cabe preguntarse por los efectos económicos que esto traería.

Si existieran los coches autónomos habría muchas profesiones que desaparecerían con el tiempo: chóferes, conductores de autobús, taxistas,

camioneros… todo lo que sea moverse por carretera sería susceptible de pasar a ser autónomo, y el primer sitio donde lo veríamos sería precisamente en el sector profesional, porque para una empresa, el ahorro de costes sería muy alto.

¿Realmente sería necesario que una persona, que sepa conducir, esté dentro del habitáculo? Al principio sí, pero con el tiempo ya no. Otro cambio económico que veríamos es la productividad.

En las grandes ciudades, llenas de atascos, tenemos a miles de personas que pierden horas y horas, en ellos, sobre todo en las llamadas "*horas punta*". Aparte de ser estresante estamos dedicando un tiempo muy valioso a una tarea muy poco productiva.

Con la llegada de los coches autónomos no podemos evitar los atascos, seguirían existiendo, incluso más, la gente que no puede permitirse un vehículo en propiedad, a lo mejor sí puede permitirse esporádicamente ir en taxi autónomo.

Pero en los atascos ya no estaremos pendientes de frenar y acelerar, sino que podremos estar haciendo otras cosas, ¿adelantar trabajo? ¿realizar llamadas? o simplemente leer el periódico.

Es difícil saberlo y generalizar, pero podemos apuntar a un aumento de la productividad, mucha gente ganaría horas al día. Otro de los motivos, por los que los coches autónomos van a ser efectivos, es la menor tasa de accidentes.

Los ordenadores son capaces de reaccionar en milisegundos ante un problema, y aunque no son creativos como los humanos, los coches autónomos van a realizar la acción correcta, con mayor rapidez que nosotros, disminuyendo de esta forma la tasa de accidentes.

Según se vayan imponiendo los coches autónomos (ya sea por preferencia de los consumidores o por legislación) habrá menos muertes y lesionados en las carreteras. Beneficiando a su vez la reducción en gasto sanitario por este concepto.

Los vehículos autónomos van a llegar en los próximos años y con ellos un gran cambio social y económico.

Los cambios van a ser profundos, significativos, pero graduales.

6. Planificación

En este apartado, se indica y detalla la planificación que se ha llevado a cabo a lo largo de la realización del proyecto. Para ello, se realizará un desglose del proyecto en cada una de sus fases de desarrollo, indicando cuantas horas se deberá de dedicar a cada una de ellas, y se incluirá un diagrama de Gantt para ilustrarlo de forma gráfica.

6.1. Desglose por fases de desarrollo

En este desglose, se mostrará por cada fase del desarrollo, la fecha de inicio, la fecha de finalización y el número de días dedicados.

Esta planificación comprende desde el día que se presentó el proyecto al tutor del mismo, hasta el día que se finalizó la documentación.

Actividad	Fecha de inicio	Fecha finalización	Días de Duración
Propuesta proyecto	28/11/2016	28/11/2016	1
Estudio de viabilidad	01/06/2017	07/06/2017	7
Análisis	08/06/2017	24/06/2017	17
Diseño	25/06/2017	04/07/2017	10
Implementación	05/07/2017	17/08/2017	40
Pruebas	18/08/2017	31/08/2017	14
Documentación	1/09/2017	25/09/2017	26

Tabla 153 - Panificación - Desglose por fases de desarrollo

6.2. Diagrama de Gantt

En este apartado se realiza la representación gráfica, mediante un diagrama de Gantt, del desglose por fases de desarrollo realizado en el apartado anterior. Se excluye de este diagrama la fase “Propuesta proyecto” por su corta duración y separación en el tiempo con el resto de fases.

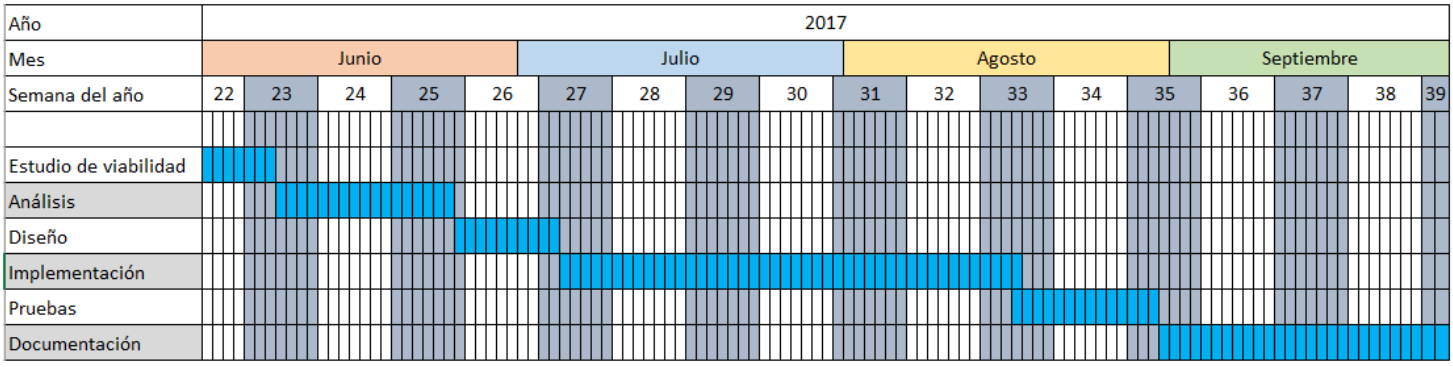

Ilustración 81 - Planificación - Diagrama de Gantt

7. Presupuesto

En este apartado, se muestra un presupuesto donde se detallan los costes estimados de realización del proyecto. A continuación, se especificarán los costes producidos por personal, hardware, software, y, por último, un resumen del presupuesto con los costes totales de la realización del mismo.

7.1. Tiempo dedicado al proyecto

Para obtener los costes, previamente se realiza un cálculo de los días y horas totales necesarias para llevar a cabo el proyecto, según lo especificado en la planificación y considerando que se trabaja a jornada completa, es decir, 8 horas diarias.

Fase del proyecto	Días	Horas
Propuesta proyecto	1	8
Estudio de viabilidad	7	56
Análisis	17	136
Diseño	10	80
Implementación	40	320
Pruebas	14	112
Documentación	26	208
	Total	**920 h**

Tabla 154 - Presupuesto - Días y horas realización proyecto

7.2. Costes de personal

En este apartado se indican los costes generados por la contratación del personal a cargo de la realización del proyecto.

En este caso, el único miembro del personal es el alumno que realiza el proyecto, por lo que se considerará que la hora de trabajo realizada por el alumno, tiene el mismo coste que la de un recién titulado en ingeniería informática *[B29]*.

Sueldo bruto anual	18000€/año
Sueldo bruto mensual	1500€/mes
Días trabajados al mes	20 días
Horas mensuales	160 h
Coste bruto hora trabajada	9,375€/h

Tabla 155 - Presupuesto - Nómina recién titulado en ingeniería informática

Conociendo los datos anteriormente citados, se realiza el cálculo del coste total de personal:

Empleado	Coste/Hora	Horas	Coste/Empleado
Luis Alberto Pantaleón del Puerto	9,375€/hora	920	8625€
		Total	**8625€**

Tabla 156 - Presupuesto - Coste total empleados

7.3. Costes de hardware

En este apartado se definen los gastos en hardware que han sido necesarios realizar para llevar a cabo el proyecto.

No se tiene en cuenta que se dispusiera de alguno de los dispositivos con anterioridad.

Componente	Precio/unidad	Unidades	Amortización	Uso	Coste proyecto
Ordenador de sobremesa	1100€	1	12 meses	4 meses	366,66€
Ordenador portátil Sony Vaio SVE1512E1EW	430€	1	12 meses	4 meses	143,33€
Xiaomi Redmi note 3 pro Special edition	195€	1	6 meses	4 meses	130€
Raspberry Pi 3 Modelo B	36,53€	1	6 meses	4 meses	24,35€
Arduino UNO rev3	21,00€	1	6 meses	4 meses	16€
Adafruit Motor/Stepper /Servo Shield v1.2	4€	1	4 meses	4 meses	4€
				Total	**684,34€**

Tabla 157 - Presupuesto - Coste total hardware

7.4. Costes de software

En este apartado se definen los gastos en licencias de software que han sido necesarios para llevar a cabo este proyecto.

No se tiene en cuenta que se dispusiera con anterioridad de la licencia de utilización de alguno de los programas. No se incluyen las licencias gratuitas.

Software	Precio/unidad	Unidades	Coste total
Microsoft Windows 10 Pro	279€	2	558€
Adobe Photoshop CS5	250€	1	250€
Microsoft Office Hogar y Estudiantes 2016	149€	1	149€
		Total	**957€**

Tabla 158 - Presupuesto - Coste total software

7.5. Resumen del presupuesto

En este apartado se mostrará un resumen del presupuesto del proyecto, este incluirá los costes totales anteriormente calculados, el coste del porcentaje de riesgo, el coste del IVA y el coste total del proyecto con la inclusión del IVA y sin él.

El porcentaje de riesgo será 15% adicional del coste total del proyecto sin IVA. Este porcentaje se utiliza para cubrir los posibles riesgos que puedan suceder y suplir aquellos gastos no queden contemplados en el presupuesto.

Concepto	Total
Costes de personal	8625€
Costes de hardware	684,34€
Costes de software	957€
Subtotal 1 (Sin riego ni IVA)	10266,34€
Riesgo (15%)	1539.95€
IVA (21%)	2479.65€
Total (sin IVA)	**11.806.29€**
Total (21% IVA)	**14.285,61€**

Tabla 159 - Presupuesto - Resumen del presupuesto

El coste total del proyecto sin IVA es de 11.806.29€, al aplicarle el 21% de IVA vigente en España, asciende a **14.285,61€.**

8. Conclusiones y trabajos futuros.

En este punto, se indicarán las conclusiones obtenidas, tanto a nivel de proyecto como a nivel personal una vez realizado el mismo.

Observando en el primer caso, que objetivos iniciales se han superado, los que no se han podido llevar a cabo y los que se han añadido.

En el segundo caso, se indicará las conclusiones personales a las que se ha llegado, durante la realización del proyecto.

Por último, se indicará que mejoras se podrían realizar sobre el proyecto, pero, por falta de tiempo, de recursos u otros impedimentos, no se han podido realizar.

8.1. Conclusiones del proyecto

Teniendo en cuenta los objetivos establecidos en 1.3 OBJETIVOS, se llevan a cabo la obtención de las conclusiones, con relación a la consecución de los objetivos marcados:

- Se ha cumplido el objetivo principal. El desarrollo de una **aplicación que permita a un usuario, el control remoto y autónomo de un vehículo, a través de la aplicación móvil.** Así como la implementación de todos sus componentes.
- **Se ha desarrollado el componente Aplicación móvil.** Esta se utiliza como interfaz para que el usuario pueda interactuar con el resto de la aplicación. Ofrece la posibilidad de seleccionar entre los dos modos de control, mostrando la interfaz correspondiente para cada uno de estos modos. La aplicación, se ha desarrollado para dispositivos Android. En este componente se han desarrollado a su vez, los siguientes sub-objetivos:
    - Como se indica en 4.3 DISEÑO DE INTERFACES, este ofrece una interfaz dedicada a cada modo de control. Además, ha sido diseñada para ofrecer al usuario una interfaz clara e intuitiva.
    - Mediante *Webview's* en la interfaz gráfica de la *Aplicación móvil*, se pueden observar las imágenes transmitidas por el *Servidor* a través de la cámara conectada al vehículo, mientras se realiza la conducción autónoma o manual de este. Estos *Webview's* se conectan a una dirección *http* donde se transmiten dichas imágenes por el *Servidor*
    - Implementa una base de datos *SQLite* para el almacenamiento de trayectos. Además, se ofrecen interfaces gráficas en *la Aplicación móvil* para poder realizar de forma simple la creación, borrado y modificación de estos trayectos, sin que el usuario conozca que se utiliza una Base de datos.
- **Se ha desarrollado el componente Servidor.** Realiza las funciones de capa intermedia entre el *Controlador* y la *Aplicación móvil*, en modo *Control*

manual. En modo de *Control automático*, realiza la toma de decisiones de la conducción autónoma. Este se ha implementado en C++ y se ejecuta sobre el vehículo en una Raspberry Pi 3 Modelo B. En este componente se han llevado a cabo los siguientes sub-objetivos:
    - Como se acaba de comentar, el hardware sobre el que se ejecuta es una Raspberry Pi 3 Modelo B, que soporta la conexión de una cámara, por lo que tendrá conectada una.
    - Puede capturar, transmitir y almacenar imágenes. Para ello utiliza la funcionalidad instalable en SO's Linux denominada *Motion*. Este permite realizar:
        - La captura continua de imágenes y su almacenamiento.
        - Levantar un servidor para la transferencia de imágenes a una dirección de tipo *http*.
    - Puede gestionar y procesar las imágenes, para ello utiliza la librería gratuita de tratamiento de imágenes *OpenCV*.
    - La toma de las decisiones de la conducción autónoma, del modo de *Control automatico,* se realiza mediante las imágenes capturadas y almacenadas por *Motion.*
    - Solo se permite la conexión de un usuario. El resto de conexiones son ignoradas hasta que finalice las peticiones del usuario de la *Aplicación móvil* que se encuentre conectado.
- **Se ha desarrollado el componente Controlador:** Se encarga de interactuar con los actuadores. Este se ha implementado en un *sketch* de Arduino y se ejecuta sobre un Arduino UNO + el *Shield* Adafruit Moto/Stepper/Servo Shield v1.2. En este componente se han llevado a cabo los siguientes sub-objetivos:
    - Ofrece una interfaz de peticiones y respuesta para la realización de movimientos, la cual, puede ser usada por el modo de *Control manual* y *Automático*. Este componente, reaccionará igual a las peticiones de los dos modos de control.
- No es necesario el reinicio, de ningún componente de la aplicación, para cambiar entre los modos de control ofrecidos. La *Aplicación móvil* será la encargada de notificar los cambios de modo de control.
- Se realiza control de errores en todos los componentes, para que el sistema sea tolerante, a los distintos errores que pueda realizar el usuario utilizando la *Aplicación móvil* o por problemas de red.

Con esto, se puede concluir que todos los objetivos establecidos se han cumplido de forma correcta. Ofreciendo al final del proyecto, un prototipo de aplicación que cumple con todos los objetivos inicialmente marcados.

La mayoría de los problemas encontrados, han sido por falta de rendimiento, teniendo en cuenta que la Raspberry Pi tiene limitaciones de potencia hardware. Respecto a la implementación, no se han encontrado problemas destacables, entrando su nivel de dificultad dentro de lo aceptado.

8.2. Conclusiones personales

Escogí realizar este proyecto, como se indicó en el apartado 1.1 MOTIVACIÓN me surgió la inquietud de saber y comprobar, si con los conocimientos adquiridos durante el grado, podría realizar un proyecto de estas características.

Ahora, una vez completado el proyecto, puedo concluir que, con los conocimientos adquiridos durante el grado, se puede llevar a cabo sin grandes dificultades.

Aunque, eso no quiere decir, que el proyecto haya sido fácil o sencillo. He tenido que invertir una gran cantidad de horas para poder ser llevado a cabo, recopilando información, contrastando la misma, a su vez, sin olvidar los estudios y amplia variedad de conocimientos en distintas plataformas y ámbitos, para la obtención de la optimización de resultados.

El proyecto, tiene un amplio grado de libertad, permitiéndome realizar la implementación que consideré oportuna en cada momento. No es el tipo de proyecto que uno está acostumbrado a realizar durante el grado. En los proyectos durante la realización de grado, se suelen establecer unas pautas que cierran el diseño a pocas posibilidades.

En cuanto al hardware utilizado, ha sido muy interesante integrar en una única aplicación, programas ejecutados en diversos dispositivos con características muy diferentes.

Como resumen final, el proyecto me ha parecido muy interesante y divertido. A su vez, me ha permitido adquirir, aplicar y desarrollar los conocimientos adquiridos respecto a la plataforma Rasberry Pi – esta no se había utilizado con anterioridad- para ningún proyecto. Por otro lado, en este proyecto se han unido tres de mis aficiones favoritas: Informática, electrónica y automovilismo. Asegurando así, una motivación extra, para la dedicación, esfuerzo y superación, minimizando los obstáculos y adversidades, que iban surgiendo en el desarrollo del proyecto. Por último, me ha permitido integrar en una única aplicación, mis conocimientos en Java, C y C++.

8.3. Trabajos futuros

El proyecto es un prototipo inicial respecto a la aplicación, la cual se quería realizar en un primero momento que, por falta de tiempo, complejidad, coste u otros, inconvenientes, no se ha podido llevar acabo.

La aplicación, sobre la idea original, no difiere en exceso. Con relación a la idea original, esta mostraba en un principio, apartados tal vez algo más llamativos.

A continuación, se indicará por cada uno de los componentes que conforman la aplicación, aquellas características que se pretendían realizar inicialmente, pero no se han podido llevar acabo, esperando que en un futuro se pueda continuar este proyecto y realizarlas:

8.3.1. Aplicación móvil

AlertDialog de conexión al servidor

En aquellas pantallas que se debe realizar una conexión con el servidor, el cuadro de dialogo generado y mostrado, no se utilizaría para especificar una IP.

En el cuadro de dialogo, debía de aparecer un listado con un conjunto de dispositivos. Estos conjuntos dispositivos, correspondería a los dispositivos que se encuentra conectado a la misma red local, que el dispositivo móvil que ejecuta la *Aplicación móvil*.

Por lo tanto, solo era necesario pulsar un elemento del listado del cuadro de dialogo, para intentar conectarse a él.

Además, en este cuadro de dialogo, se dispondría de posiciones preferentes para aquellos dispositivos a los que se hubiera realizado una conexión con anterioridad.

Interfaz *Control manual*

En la pantalla de *Control manual*, no se utilizarían botones para realizar peticiones de movimiento del vehículo o de la cámara.

En el caso original, se debían de utilizar joystick, que, según su desplazamiento, permitieran regular el movimiento que se quería realizar, controlando la velocidad de dicho movimiento.

8.3.2. Servidor

Movimientos conducción autónoma

La conducción autónoma, no se realizaría mediante movimientos discretos, sino que inicialmente eran continuos.

En este caso, no se pudo llevar a cabo esto, debido a que por el momento no se puede asegurar un *framerate* minino de imágenes capturadas por la cámara. Esto provoca, que el vehículo en determinados casos, se mueva más rápido que la captura de imágenes, lo que provoca constantes salidas del circuito.

Giros conducción autónoma

Los giros de la conducción autónoma no serían únicamente movimientos para girar a la izquierda o a la derecha sin especificar un ángulo. Estos giros, se realizarían especificando ángulos absolutos de giro, los cuales el vehículo realizaría.

Detector de señales

Inicialmente, el *Servidor* iba a disponer de un detector de señales. Este detector, reconocería señales que aparecieran en las imágenes capturadas por la cámara del vehículo y la distancia a la que se encuentran.

Una vez detectada la señal, se comprobaría la distancia a la que se encuentra el vehículo de dicha señal. Si el vehículo se encuentra a suficiente distancia, el *Servidor* llevaría a cabo las acciones correspondientes del control del vehículo, para cumplir la señal.

Por ejemplo, si observara una señal de STOP, cuando este se encontrará lo suficientemente cerca, detendría el vehículo durante unos segundos.

8.3.3. Controlador

Detector de colisiones

El vehículo, dispondría de un sensor de ultrasonidos utilizado para detectar posibles colisiones u obstáculos. El *controlador*, sería el encargado de obtener del sensor, la información.

Si se detecta que es posible una colisión, el *Controlador* detendría el vehículo y lo notificaría, mediante el puerto serial.

Este procesamiento, lo debe de realizar el Controlador para llevar a cabo una acción lo más rápido posible, evitando la colisión.

9. Acrónimos, abreviaturas y definiciones.

9.1. Acrónimos y abreviaturas

[A1] IDE. Integrated Development Enviroment
[A2] SDK. Software Development Kit
[A3] GNU GPL: GNU General Public License
[A4] SO. Sistema operativo
[A5] RAM. Random Memory Acess
[A6] CPU. Central Processing Unit
[A7] GPU. Graphics Processor Unit
[A8] PPI. Pixels Per Inch
[A9] LCD. Liquid Cristal Display
[A10] IPS. IN-Plane Switching
[A11] TFT. Thin Film Transistor
[A12] mAh. Mili amperios hora
[A13] GHz. Gigahercios
[A14] SBC. Single Board Computer
[A15] SOC. Sistem On a Chip
[A16] HDMI. High-Definition Multimedia Interface
[A17] GPIO. *General Purpose Input/Output*
[A18] SD. Secure Digital
[A19] MMC. Multimedia Card
[A20] JTAG. Joint Test Action Group
[A21] A. Amperios
[A22] mA. Mili amperios
[A23] V. Voltios
[A24] PC. Personal Computer
[A25] IP. Internet Protocol
[A26] BBDD. Base de Datos.
[A27] HTTP. Hypertext Transfer Protocol
[A28] IVA. Impuesto sobre el Valor Añadido
[A29] LIDAR. Laser Imaging Detection and Ranging
[A30] BN. Blanco y Negro

9.2. Definiciones

[D1] **ActionBar**(Android). Es una barra persistente que se establece en la parte superior de las pantallas de las aplicaciones. Ejemplo:

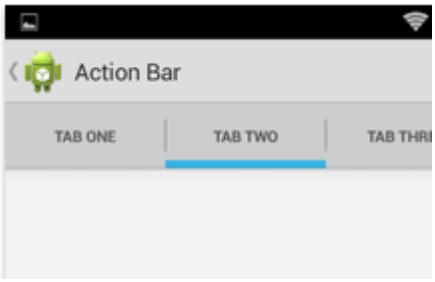

Ilustración 82 - Ejemplo de actionBar

[D2] ***Button back*** (Dispositivos móviles)**.** Botón físico o virtual del que disponen la mayoría de dispositivos móviles utilizado para volver "atrás". Ejemplos:

Ilustración 83 - Ejemplo button back físico

Ilustración 84 - Ejemplo button back virtual

[D3] **Sketch** (Arduino)**.** Es el nombre que se le da a los programas para placas Arduino.

[D4] **Background** (Android)**.** Hace referencia al color de fondo que tiene cada elemento que compone una aplicación.

[D5] **ToggleButton**(Android). Botón que dispone de dos posiciones *isChecked* y *unChecked.* Ejemplo:

Ilustración 85 - Ejemplo ToggleButton en estado unChecked

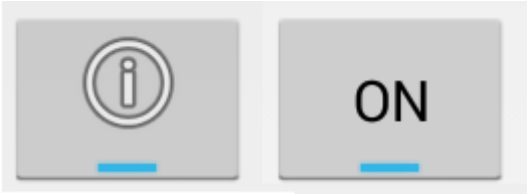

Ilustración 86 - Ejemplo ToggleButton en estado isChecked

[D6] Notificación tipo Toast. Es un tipo de notificación de Android no persistente**.** Ejemplo:

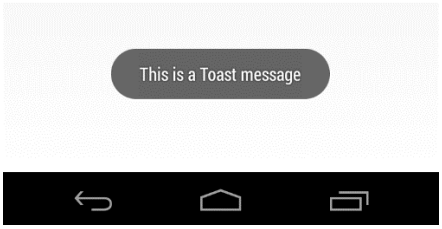

Ilustración 87 - Ejemplo notificación de tipo Toast

[D7] AlertDialog. Es un tipo de cuadro de dialogo de Android, este se mantiene persistente hasta que el usuario selecciona un botón de los que dispone o pulsa fuera del cuadro de dialogo.

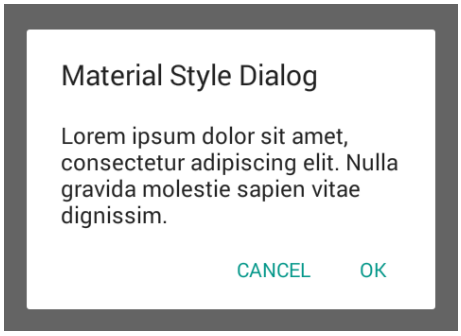

Ilustración 88 - Ejemplo cuadro de dialogo AlertDialog

[D8] Socket. Abstracción software por el cual dos programas pueden establecer una comunicación y realizar un intercambio de datos [B36].

[D9] Timertask(Android)**.** Recurso software del que dispone Android, en el cual se puede especificar la realización de una tarea pasado un tiempo especificado.

10. Bibliografía

[B1] Sistema operativo móvil Android. *Septiembre 2017*

https://es.wikipedia.org/wiki/Android

[B2] Sistema operativo móvil iOS. *Septiembre 2017*

https://es.wikipedia.org/wiki/IOS

[B3] Publicar aplicaciones en Google Play. *Septiembre 2017*

https://cincodias.elpais.com/cincodias/2015/02/01/lifestyle/1422792260_243066.html

[B4] Publicar aplicaciones en Amazon AppStore. *Septiembre 2017*

http://www.consultor-seo.com/como-publicar-tus-aplicaciones-moviles-en-amazon/

[B5] Lenguaje de programación Swift. *Septiembre 2017*

https://es.wikipedia.org/wiki/Swift_(lenguaje_de_programaci%C3%B3n)

[B6] Licencias de desarrollo iOS y publicación de aplicaciones AppleStore. *Septiembre 2017*

http://www.migueldiazrubio.com/desarrollo-ios-tipos-de-licencias-de-desarrollo/

[B7] Características dispositivo móvil Xiaomi Redmi Note 3 Pro Special Edition. *Septiembre 2017*

https://www.kimovil.com/es/donde-comprar-xiaomi-redmi-note-3-prospecial-edition-2gb-16gb

[B8] Características dispositivo móvil Xiaomi Redmi 2. *Septiembre 2017*

https://www.kimovil.com/es/comparar-moviles/name.xiaomi%20redmi%202

[B9] Características dispositivo Alcatel One Touch Pop C7. *Septiembre 2017*

https://www.kimovil.com/es/donde-comprar-alcatel-onetouch-pop-c7

[B10] Características Vodafone Smart Speed 6. *Septiembre 2017*

http://www.gsmarena.com/vodafone_smart_speed_6-7608.php

[B11] Que son y características de las placas SBC. *Septiembre 2017*

https://www.hwlibre.com/que-es-una-placa-sbc/

https://es.wikipedia.org/wiki/Placa_computadora

[B11] Historia y características placas SBC Raspberry Pi. *Septiembre 2017*

https://es.wikipedia.org/wiki/Raspberry_Pi

https://www.raspberrypi.org/

http://computerhoy.com/noticias/hardware/que-es-raspberry-pi-donde-comprarla-como-usarla-8614

[B12] Coste Raspberry Pi Modelo A+ en distribuidor oficial. *Septiembre 2017*

http://es.rs-online.com/web/p/processor-microcontroller-development-kits/8332699/?src=raspberrypi

[B13] Coste Raspberry Pi Modelo B+ en distribuidor oficial. *Septiembre 2017*

http://es.rs-online.com/web/p/processor-microcontroller-development-kits/8111284/?src=raspberrypi

[B14] Coste Raspberry Pi 2 Modelo B en distribuidor oficial. *Septiembre 2017*

http://es.rs-online.com/web/p/processor-microcontroller-development-kits/1259525/?src=raspberrypi

[B15] Coste Raspberry Pi Zero en distribuidor oficial. *Septiembre 2017*

https://www.kubii.fr/es/raspberry-pi-3-2-b/1401-raspberry-pi-zero--3272496003408.html

[B16] Coste Raspberry Pi Zero W en distribuidor oficial. *Septiembre 2017*

https://www.kubii.fr/es/pi-zero-w/1851-raspberry-pi-zero-w-3272496006997.html

[B17] Coste Raspberry Pi 3 Modelo B en distribuidor oficial. *Septiembre 2017*

http://es.rs-online.com/web/p/processor-microcontroller-development-kits/8968660/?src=raspberrypi

[B18] Documentación procesador Broadcom BCM2835. *Septiembre 2017*

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md

[B19] Documentación procesador Broadcom BCM2836. *Septiembre 2017*

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/README.md

[B20] Documentación procesador Broadcom BCM2837. *Septiembre 2017*

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md

[B21] Historia y características placas SBC BeagleBoard. *Septiembre 2017*

https://es.wikipedia.org/wiki/BeagleBoard

https://beagleboard.org/

[B22] Especificaciones técnicas placa BeagleBoard. *Septiembre 2017*

http://beagleboard.org/beagleboard

[B23] Especificaciones técnicas placa BeagleBoard-xM. *Septiembre 2017*

http://beagleboard.org/beagleboard-xm

[B24] Especificaciones técnicas placa BeagleBone. *Septiembre 2017*

http://beagleboard.org/bone-original

[B25] Especificaciones técnicas placa BeagleBone Black. *Septiembre 2017*

http://beagleboard.org/bone

[B26] Coste BeagleBoard-xM en distribuidor oficial. *Septiembre 2017*

http://www.mouser.es/ProductDetail/BeagleBoard/PTI-BEAGLE_BOARD_XM_KIT/?qs=eFOtW6xYNWEdeOUjxRp8gQ%3d%3d

[B27] Coste BeagleBone Black en distribuidor oficial. *Septiembre 2017*

http://eu.mouser.com/search/ProductDetail.aspx?R=0virtualkey0virtualkeyBBB01-SC-505

[B28] Que es el Open-source Hardware y Adafruit. *Septiembre 2017*

https://en.wikipedia.org/wiki/Adafruit_Industries

https://en.wikipedia.org/wiki/Open-source_hardware

[B29] Características técnicas Adafruit DC & Stepper Motor HAT. *Septiembre 2017*

https://www.adafruit.com/product/2348

[B30] Historia y características de las placas Arduino. *Septiembre 2017*

https://es.wikipedia.org/wiki/Arduino

[B31] Características técnicas Arduino UNO rev.3. *Septiembre 2017*

http://arduino.cl/arduino-uno/

[B32] Características técnicas Adafruit Motor/Stepper/Servo Shield for Arduino v1.2. *Septiembre 2017*

https://www.adafruit.com/product/81

[B33] Características técnicas Adafruit Motor/Stepper/Servo Shield for Arduino v2. *Septiembre 2017*

https://www.adafruit.com/product/1438

[B34] Características técnicas ATmega328. *Septiembre 2017*

http://www.microchip.com/wwwproducts/en/ATmega328

[B35] Coste Adafruit Motor/Stepper/Servo Shield for Arduino v1.2 no oficial. *Septiembre 2017*

http://www.ebay.es/itm/L293D-Motor-Drive-Shield-Expansion-Board-Arduino-UNO-Mega-Controlador-Servo-R001-/201577860897?hash=item2eeefa0f21:g:d28AAOSwmgJY7zIG

[B36] Historia y características de Python. *Septiembre 2017*

https://es.wikipedia.org/wiki/Python

http://www.cuatrorios.org/index.php?option=com_content&view=article&id=161:principales-caracteristicas-del-lenguaje-python&catid=39:blogsfeeds

[B37] Historia y características de C++. *Septiembre 2017*

https://es.wikipedia.org/wiki/C%2B%2B

[B38] Información de sockets en internet. *Septiembre 2017*

https://es.wikipedia.org/wiki/Socket_de_Internet

[B39] Información salarios recién titulados en informática en España. *Septiembre 2017*

https://www.infojobs.net/

https://www.xataka.com/tecnologiazen/la-realidad-del-perfil-de-informatico-junior-en-espana-segun-los-informes

[B40] Funcionamiento "autopilot" Tesla. *Septiembre 2017*

https://www.mediatrends.es/a/119798/autopilot-coche-tesla-conduccion-autonoma-piloto-automatico-para-coches-accidente-elon-musk/

https://www.tesla.com/es_ES/autopilot

[B41] Funcionamiento coche autónomo Waymo. *Septiembre 2017*

https://www.motorpasion.com/coches-hibridos-alternativos/como-funciona-el-coche-autonomo-de-google

[B42] La arquitectura cliente-servidor. *Septiembre 2017*

https://es.wikipedia.org/wiki/Cliente-servidor

https://laurmolina7821.wordpress.com/1-1-3-aplicaciones-de-2-3-y-n-capas/

[B43] El patrón de diseño modelo-vista-controlador. *Septiembre 2017*

https://book.cakephp.org/2.0/es/cakephp-overview/understanding-model-view-controller.html

[B44] Guía de diseño *Material desing* de Google. *Septiembre 2017*

https://material.io/

[B45] *Material desing* apartado dedicado a *Launch screens*. *Septiembre 2017*

https://material.io/guidelines/patterns/launch-screens.html#launch-screens-types-of-launch-screens

[B46] Documentación SQLiteOpenHelper. *Septiembre 2017*

https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[B47] Documentación librería Servo de Arduino. *Septiembre 2017*

https://www.arduino.cc/en/Reference/Servo

[B48] Documentación librería AFmotor de AdaFruit. *Septiembre 2017*

https://learn.adafruit.com/adafruit-motor-shield/overview

[B49] Familia de direcciones AF_INET. *Septiembre 2017*

https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzab6/cafinet.htm

[B50] Documentación librería WiringPI. *Septiembre 2017*

http://wiringpi.com/

[B51] Funcionalidad Motion de Linux. *Septiembre 2017*

https://github.com/Motion-Project/motion

[B52] Documentación librería OpenCV. *Septiembre 2017*

http://opencv.org/

[B53] Filtrado de tipo OTSU. *Septiembre 2017*

https://en.wikipedia.org/wiki/Otsu%27s_method

[B54] DGT vehículos autónomos. *Septiembre 2017*

http://www.dgt.es/es/prensa/notas-de-prensa/2015/20151116-trafico-establece-marco-realizacion-pruebas-vehiculos-conduccion-automatizada-vias-abiertas-circulacion.shtml

[B55] Documentación oficial LOPD. *Septiembre 2017*

http://www.agpd.es/portalwebAGPD/canaldocumentacion/informes_juridicos/reglamento_lopd/index-ides-idphp.php

[B56] Información licencias software libre GNU GPL y BSD. *Septiembre 2017*

http://biblioteca.uclm.es/Archivos/Investigacion/Software_libre.pdf

[B57] Ejemplo modelo-vista-controlador

http://blog.cubenube.com/2011/11/la-arquitectura-modelo-vista.html

[B58] Debate Vehículo Autónomo Universidad de Deusto. *Septiembre 2017*

http://www.deusto.es/cs/Satellite/deusto/es/universidad-deusto/vive-deusto/el-coche-del-futuro-a-debate-en-un-acto-conmemorativo-del-40-aniversario-de-la-facultad-de-ingenieria/noticia.